SNAVA - A Real-Time Multi-FPGA Multi-Model Spiking Neural Network Simulation Architecture

Athul Sripad[b], Giovanny Sanchez[a,*], Mireya Zapata[b], Vito Pirrone[b], Taho Dorta[b], Salvatore Cambria[b], Albert Marti[b], Karthikeyan Krishnamourthy[b], Jordi Madrenas[b]

[a] *Instituto Politecnico Nacional, ESIME Culhuacan, Av. Santa Ana N 1000, Coyoacan, 04260, Distrito Federal, Mexico*
[b] *Dept. of Electronics Engineering, Universitat Politècnica de Catalunya, Jordi Girona, 1-3, edif. C4, 08034 Barcelona, Catalunya, Spain*

Abstract

Spiking Neural Networks (SNN) for Versatile Applications (SNAVA) simulation platform is a scalable and programmable parallel architecture that supports real-time, large-scale, multi-model SNN computation. This parallel architecture is implemented in modern Field-Programmable Gate Arrays (FPGAs) devices to provide high performance execution and flexibility to support large-scale SNN models. Flexibility is defined in terms of programmability, which allows easy synapse and neuron implementation. This has been achieved by using a special-purpose Processing Elements (PEs) for computing SNNs, and analyzing and customizing the instruction set according to the processing needs to achieve maximum performance with minimum resources. The parallel architecture is interfaced with customized Graphical User Interfaces (GUIs) to configure the SNN's connectivity, to compile the neuron-synapse model and to monitor SNN's activity. Our contribution intends to provide a tool that allows to prototype SNNs faster than on CPU/GPU architectures but significantly cheaper than fabricating a customized neuromorphic chip. This could be potentially valuable to the computational neuroscience and neuromorphic engineering communities.

Keywords: Digital neural simulation, FPGA, SNNs, neuromorphic systems.

*Corresponding author
Email address: giovas666@hotmail.com (Giovanny Sanchez)

1. Introduction

In the present era, simulation of complex biological neural systems is a trending research area. Neuroscientists consider it an important tool in understanding the structure and dynamics of the human brain. The unparalleled performance of the human brain has encouraged many engineers to develop bio-inspired simulators that mimic part of the human brain functionality. Though there have been many research projects on this discipline, most of those implementations run on conventional Von Neumann computing architectures. Albeit their advantages of shorter design time and freedom in configurability, they require bulky computing systems for simulation of large-scale networks. For example, the IBM's Blue-Gene supercomputer runs the simulation of very large-scale spiking neural networks at ion-channel level [1]. However, the simulation of extreme-scale and high-complex SNN models in software-based simulators, which are implemented in Von Neumann machines, could not match the biological spiking rate (milli seconds) and demand large amount of power. For instance, the Blue-Gene consumes around 8.4 GWatts [2].

Analog implementations exploit transistor's sub-threshold range operations to create compact and high-speed processing neural simulators. "BrainScaleS" [3] is one of the most prominent projects in full custom analog design to simulate exponential integrate-and-fire neurons [4]. Analog implementations offer extremely low area and energy consumption for very large-scale networks; however, they are difficult to program and to scale. Hence, the full custom analog implementations could be useful for those applications where the behavior of the SNN is very well defined and characterized. Apart from this fact, analog implementations are very costly to fabricate and time consuming to design and to tune.

In contrast to analog-based solutions, most digital implementations are less costly and more flexible. Currently many implementations use general-purpose multiprocessors, Graphical Processing Units (GPUs) or FPGAs. These digi-

tal architectures offer wide range of flexibility and re-configurability to process large-scale SNN models at high speed.

One of the recently highlighted multiprocessor-based SNN simulators is TrueNorth [5]. TrueNorth implements Leaky Integrate-and-Fire (LIF) neurons with high number of synapses without plasticity. SpiNNaker [6] is another well-known multiprocessor-based SNN simulator. Its programmable feature allows SpiNNaker to support different SNN models at the cost of highly complex processing cores.

GPU cards can provide a powerful solution when highly parallel computing is required. Complexity of SNNs make memory management and spike propagation major obstacles in using general purpose GPUs. Several advanced GPU-based simulators have been proposed during the last ten years, one of them is NEST (Neural Simulation Tool) [7] that supports several neural & synaptic models. Its low degree of biophysical detail has been a critical issue. NeoCortical Simulator 6 (NCS6) addresses this issue and reaches good levels of biological detail [8]. NCS6 natively supports LIF and Izhikevich [9] models and also allows the user to design his/her own interface for other neural models. NeMo is one other famous GPU based SNN simulator [10] that enables users to simulate models like Izhikevich [9], integrate-and-fire models and Kuramoto oscillators [11].

FPGA-based SNN simulation has been proposed and implemented in several works [12–15]. One of the proposed works [12] is focused on building a multiple FPGA-based architecture with high speed communication by using high speed serial links available in advanced FPGA boards. However, there is no mechanism that can manage congestion in the SNN network in case of saturation. In addition, this architecture can simulate Izhikevich neurons [9] with fixed pipeline stages. This greatly reduces the capacity of the system for supporting different SNN models. Moreover, this architecture is designed for the simulation of simple and specific SNN models that do not take plasticity of synapses into consideration. This has special relevance since plasticity [16–18] is an important feature to be taken into account while considering real-time self-evolving neu-

romorphic systems, which is the target of simulators [14, 19–21]. Apart from this architecture, Zamarreño et al. [13] proposed a scalable-reconfigurable neuromorphic Address Event Representation (AER) configured as 2D mesh. They claim that the proposed architecture is capable of managing spike traffic using routing approaches in a single or multiple FPGAs. The architecture simulates simple Integrate-and-Fire (IF) neurons to perform the convolution operation [22] that is used in image processing (character recognition), but neurons do not involve plasticity. Another architecture was developed by Neil et al. [14]. They proposed a FPGA-based SNN accelerator that is called Minitaur. The aim of Minitaur is to implement a large-scale LIF neural network to efficiently perform handwritten digit classification. Part of their future work is to explore new effective learning methods to improve the accuracy of the classification. One of the recent FPGA-based systems has been proposed by Luo et al. [15] to simulate the cerebellum. Their results show that the proposed platform simulates Golgi and granular cells at high processing speeds and it could be adapted as a potential neuroprosthetic platform for future clinical applications.

The FPGA digital implementations above discussed trade off model flexible and high speed processing. The GPU and general purpose multiprocessor approaches seem to have the flexibility to implement several SNN models and the scalability to implement fairly large-scale networks. But all these implementations rely on a general purpose Instruction Set Architecture (ISA) and on chip communication to simulate SNN. Evidently, there would be some performance loss and power consumption because of certain functionalities that are useless for SNN simulation. This work presents a complete solution for real-time multi-model SNN simulation called SNAVA. The SNAVA solution has been designed by analyzing several SNN models. The ISA of this digital architecture has been tailored to SNN simulation to get the best out of the utilized hardware. Hardware-software co-design is the strength of the solution. Software to design, configure and monitor the network has been tightly coupled with the hardware for seamless operation. SNAVA is presented in this article along with the software and a functional prototype. The article has been split into eight sections.

Section 2 presents some of the neural models that were considered for the design of SNAVA leading to several design decisions. Section 3 gives an overview of the SNAVA digital architecture. Section 4 introduces the SNAVA prototype along with the software to design, configure and monitor SNN called $SNAVA\ Conf$ and SNAVA HMI. Section 5 shows the performance analyses of SNAVA and a comparison between its performance and digital embedded systems. Section 6 points out and compares the related work. Section 7 describes the simulation of SynFire Chain (SFC) algorithm to demonstrate the flexibility SNN modeling capability of the proposed architecture to solve complex non-linear models at high processing speeds. Section 8 shows a simple experiment to demonstrate the capability of SNAVA to support plasticity. Finally, Section 9 concludes with the further planned work.

2. Neuron model analysis

Several SNN models have been proposed in the last few decades. Most of them model the ion channels that are responsible for generating spikes at the axon initial segment. The classic and popular one proposed by Hodgkin and Huxley [23] describes a conductance-based neuron by reproducing electrophysiological measurements to a very high degree of accuracy. Unfortunately, this model is very complex; setting aside it is difficult to analyze, it is computationally expensive in numerical implementations. The simplest SNN model is the IF model, enhanced to the LIF, as the IF is too simple in general. The LIF model is simple to understand, easy to implement and commonly used in SNNs. It is extensively used in applications involving processing of time-varying signals [24]. Based on the LIF models there are several other adaptations with various levels of detail of neural behaviors. One such neural model was proposed by Izhikevich [9]. This model combines the biological accuracy of the Hodgkin–Huxley-type dynamics and the computational efficiency of LIF neurons to reproduce the spiking behavior of twenty neurons dynamics [9]. Another SNN model, which was proposed by Iglesias and Villa [25], focuses on including

the Spike-Time Dependent Plasticity (STDP) rule in the synapse model and the neuron is modelled as a LIF neuron with background noise [25].

By analyzing the models proposed above by various scholars, we observed that the parameters of the synapse model contribute to determine the next state of the associated neuron and the present state of the neuron parameter of the model contributes to determine the next parameter state of the associated synapses. The final goal is always to determine the membrane potential that decides when a neuron fires a spike. In addition, it can be seen that the parameters involved in modelling the neuron and the associated synapses are local to that particular neuron in most cases. Thus, each neuron can be simulated with a processor with a local memory to hold the modelling parameters as they will be accessed frequently and an array of processor blocks simulates all neurons and concerned synapses simultaneously by using the same algorithm. Therefore, they will require the same set of operations with different local parameters along with a single control unit. This allows the using of a Single Instruction Multiple Data (SIMD) processor array.

A processing flow that would suit these models for each neuron-synapse unit would be:

1. Computation of synapse parameters from the present neuron states.
2. Computation of neuron parameters from the evolved synapse states.
3. Communication of spikes, if any, between neurons.

From the above 1&2 can be grouped and called Processing phase (Phase I) and 3 can be defined as Spike distribution phase (Phase II). It can be argued that the processing of both synapses and the neurons should be simultaneous to be close to biology. But in reality, the number of synapses is greater by many magnitudes than the number of neurons. So the physically plausible option can be each neuron and associated synapses simulated by one processor and a large number of processors can simulate large neural networks, as shown in Fig. 1. Based on the previous assumption, the proposed processor can simulate neurons and synapses using a proposed flowchart (see Fig. 1) that is mainly composed

of an initialization process, load neuron parameters, a synaptic loop, a neuron computation and spike distribution process. The initialization process is linked to define the neural-synaptic parameters, connectivity and SNN algorithm. The synaptic loop can be dedicated to load synaptic parameters into the registers of the processor and process each synapse sequentially. After that, the neuron computation process determines the membrane potential (neural parameter). The last process is in charge of propagating the resulting spikes into the network. Once Phase II finishes the spike distribution, Phase I can be resumed to calculate the synaptic-neural state for the next simulation step. All the above has been taken into account in SNAVA and it has been designed to support all the SNN models that follow the conventional threshold-based spiking approach, right from the simplest IF model to the complex Iglesias and Villa with STDP [25], where the communication between neurons is only through spikes.

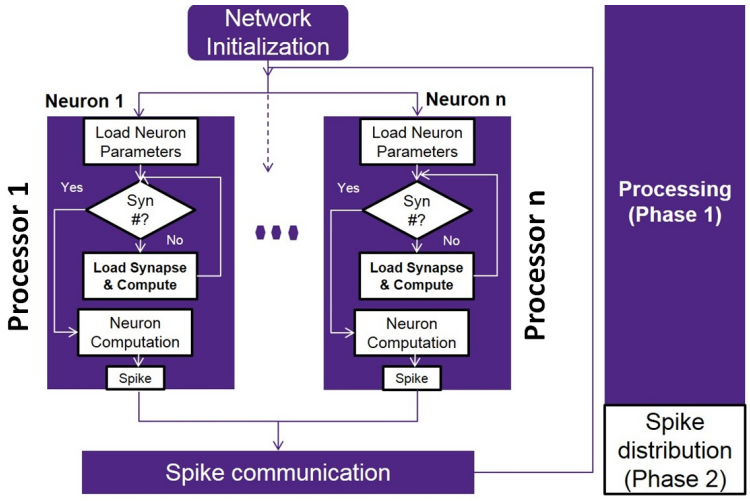


Figure 1: Execution flow for SNN simulation.

3. SNAVA architectural overview

SNAVA architecture is mainly composed of an array of SIMD units, a single control unit and a number of communication units that support software

to configure and monitor the system in real time (1 ms time step simulation). In addition, SNAVA is a scalable architecture and has the flexibility to be implemented into several chips forming a system in any topology of user's choice [26]. The design of SNAVA architecture was based on three key ideas that have contributed to simulate large-scale SNN models at high processing speeds and communication speeds. These ideas are related to the processing system, memory system and communication system.

1. Processing system/Virtualized topology: The virtualization concept also known as time-multiplexing of neural computations that increases the capacity of the architecture in order to support several neurons by using the same processor. This technique tries to minimize the consumption of hardware in exchange for the increase in the execution time. The implementation of the virtualization concept is feasible in SNAVA architecture. This is because SNAVA was designed to execute SNN models at high processing speed achieving less than 1 ms for every time step simulation. Assuming the time resolution in the biological neurons is considered to be around 1 ms. Therefore, several neurons could be simulated in that time by using the same array.
2. Memory system: A distributed memory system has been implemented in the current architecture. The memory system allows accessing the memory in each processor by spending a single clock cycle. Putting into practice of such a system was possible since modern FPGAs have thousands of Blocks of RAM integrated in them which could be used for this purpose. Besides, the BRAMs have been manufactured in such a way that they optimize the area and power consumption.
3. Communication System: The technological advancements in terms of communication have enabled the development of new protocols of communication at very high speeds. This important aspect has been considered in design of SNAVA in order to be upgradeable with the newest technology without having to make radical changes in the architecture.

The functional block schematic of SNAVA is shown in Fig. 2.

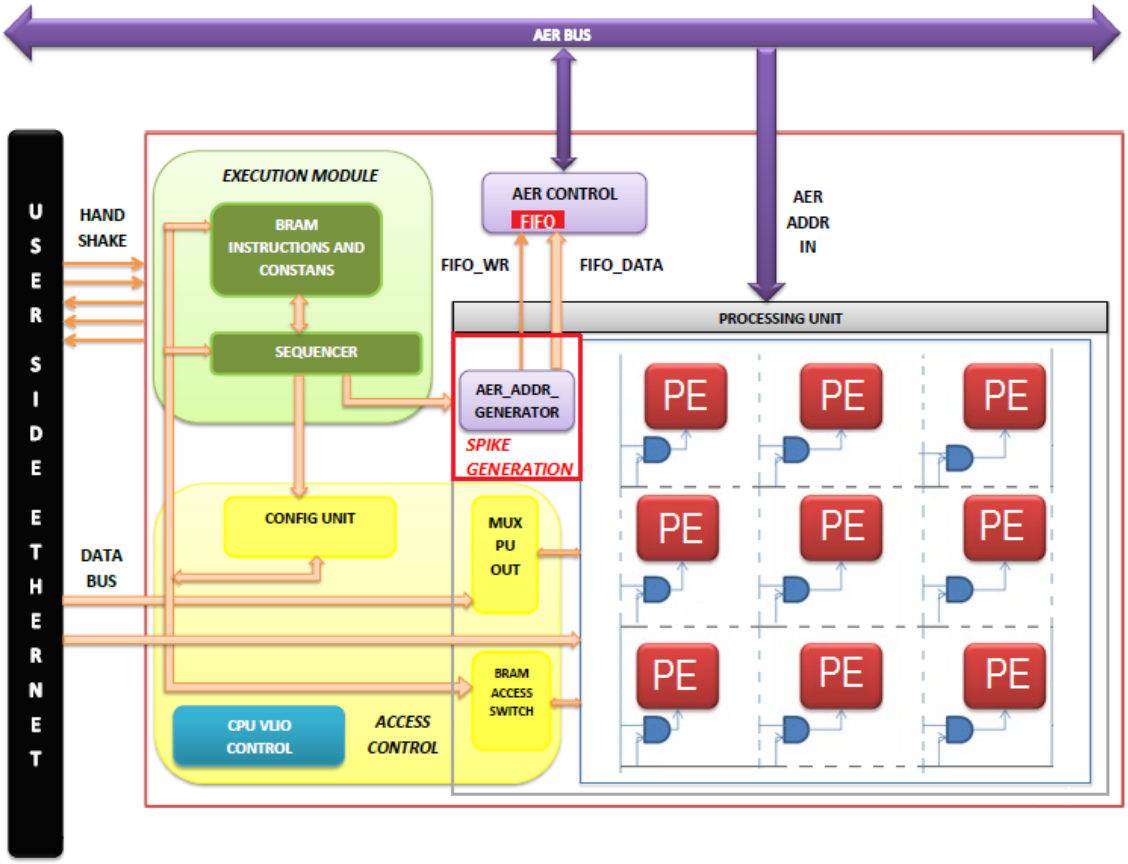


Figure 2: Architectural Overview of SNAVA.

It consists of the following main blocks.

a) **The Processing Element (PE) array** – This array computes the neuron and synapse model algorithm. The functional structure of a Central Processing Element (CPE), a PE consists of a 64-bit LFSR in Galois configuration, a Synaptic Block Random Access Memory (BRAM), Neural BRAM, a Content Addressable Memory (CAM) and a Spike register, as shown in Fig. 3. The synaptic parameters are stored in the Synaptic BRAM and neural parameters are stored in the Neural BRAM. The CAM contains the addresses of

the neurons that are connected to the PE. The spike register is an n-bit register where n is the number of synapses associated with those neurons. The Content Addressable Memory (CAM) and spike register are active during spike distribution phase. The CAM contains the addresses of the neurons that have synaptic connections with the PE it belongs to. It reads from the AER bus the broadcasted addresses of the neurons that spiked [26] and generates matches if the PE has synaptic connections to that neuron. This match is stored in the position that corresponds to the synaptic connection in the spike register. This information is used by the CPE while processing synapses.

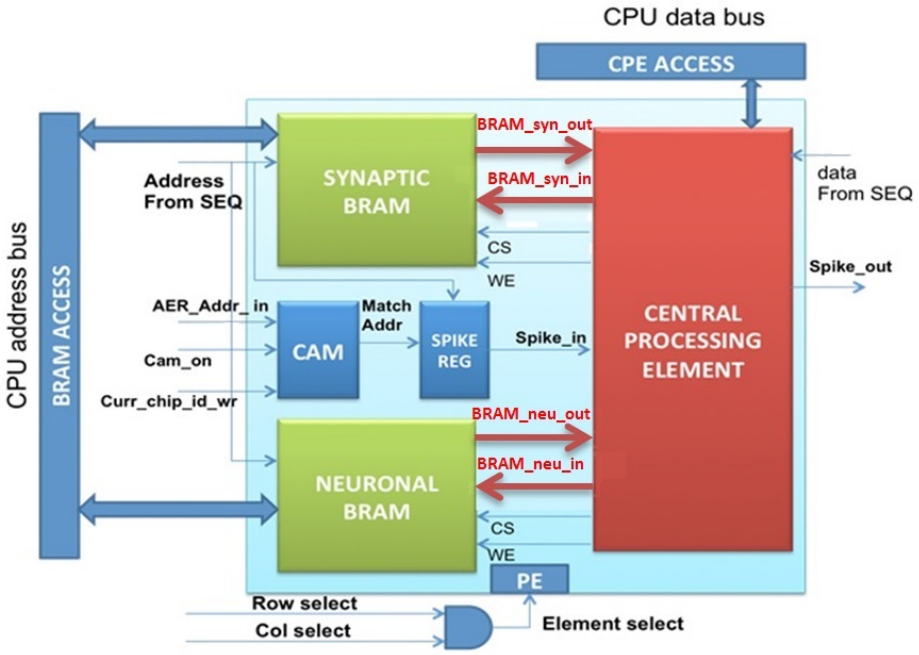


Figure 3: The structure of the Processing Element with interface to other modules.

The CPE is responsible for performing the algorithm computations. It implements arithmetic, logic, data movement and some SNN-customized operations controlled by the execution unit (see Table A1). It mainly consists

of an Arithmetic Logic Unit (ALU), an active register bank and a shadow register bank, each of 8 16-bit registers. As mentioned above, a key feature of the proposed PE is linked to use of distributed memory banks to optimize the memory access by avoiding the bottleneck that arises in many other architectures of loading/storing global parameters. In our design, the data ports of the BRAMs are wired to the active and shadow registers of the CPE respectively, as shown in Fig. 4. Therefore, the memory system allows accessing the memory in each PE by spending a single clock cycle and eliminating critical paths. In addition, the shadow register as the name implies serves as a temporary storage for the active registers providing space for complex algorithms and data move operations are possible between the active and the shadow registers either as bulk or single. The positive benefit of the shadow registers is that a single neural parameter or multiple neural parameters can be accessed in a single clock avoiding access to the BRAM memory which helps to increase the processing speed of the SNN algorithms.

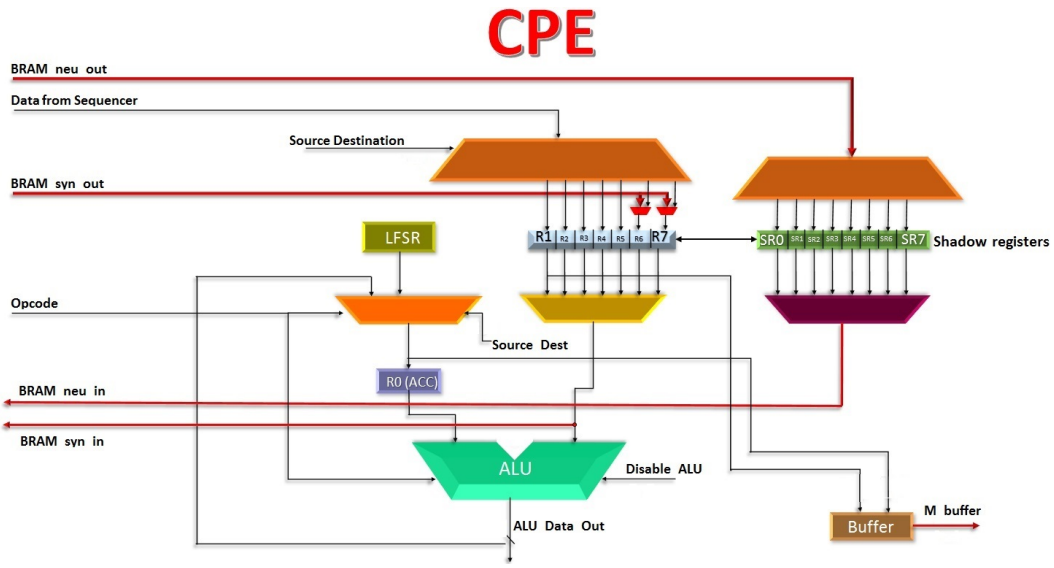


Figure 4: Central Processing Element data path.

b) **Execution module** – The Execution module consists of a sequencer (Seq)

and Block-RAM (BRAM) that stores the instructions to be executed and the global constants. The sequencer is capable of executing ALU, Data movement, conditional and unconditional control transfer, and looping instructions as given in Table A1. The data path of the sequencer consists of a program counter (PC) that points to the memory position of the next instruction. The length of the memory program is limited to 1024 words due to 10-bit addressing of the PC. At position 0, the instruction memory stores the IMEMP pointer, which indicates the position of the first program instruction. Another pointer, the DMEM indicates the position of the constants to be loaded to the registers of the PE array. Three Last-In First-Out (LIFO) stacks are provided to keep track of iterations in the LOOPS and LOOPN instructions (see Table A1). The LIFO depth by default is 8, so eight nesting levels are supported. Nesting requires storing the PC contents, the current iteration number and the loop limit in the stacks depicted in Fig. 5 (PC_LIFO, LOOP_LIFO and LOOP_LIFO2 blocks).

The sequencer is defined as 3-stage pipelined Moore-type finite state machine, to determine the control flow in the system. These 3 stages are: fetch, decode and execute. Fig. 6 shows the FSM state and works as follows: a synchronous reset initializes the state, so that all registers of the sequencer are set to 0. After the reset state, the first position of the BRAM instruction is read and loaded into the instruction address pointer IMEMP. Therefore, the program starts from the location pointed by the IMEMP. The next state is FETCH. In this state the BRAM is read using the IMEMP as base address so the first instruction is read and the opcode is decoded. The PC is incremented to point to the next instruction. This process is carried out simultaneously. While the instruction is decoded the next instruction is read by the incremented PC counter. This mechanism allows pipeline execution, only for instructions that need two clock cycles the pipeline process must break. These instructions are READMP, ENDL, GOTO and RET.

In addition, the sequencer contains data monitoring logic that is interfaced with the user-side Ethernet block that connects SNAVA to the host CPU.

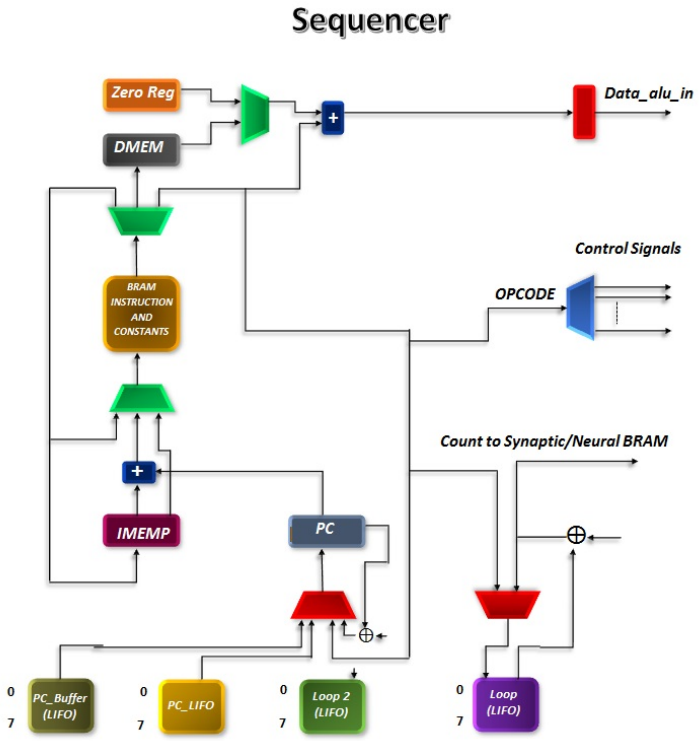


Figure 5: Sequencer data path.

It shares a few handshake signals for the transmission/reception of system information/control data. The data monitor has been implemented mainly to provide user with a control to send neural and synaptic information to the Host CPU through the user-side Ethernet communication block for monitoring the network at any point in the algorithm execution during the synaptic-neural processing phase. With different patterns of STOREB and HALT instructions (see Table A1), the user can initiate different modes of monitoring. There are two modes in which the data monitor logic operates, namely on-line scan and off-line scan. The STOREB instruction stores the synapse/neural data from certain active registers as decided by the user to a monitor buffer inside every CPE (see Fig. 4).

(a) When a STOREB instruction is followed by HALT, the sequencer halts

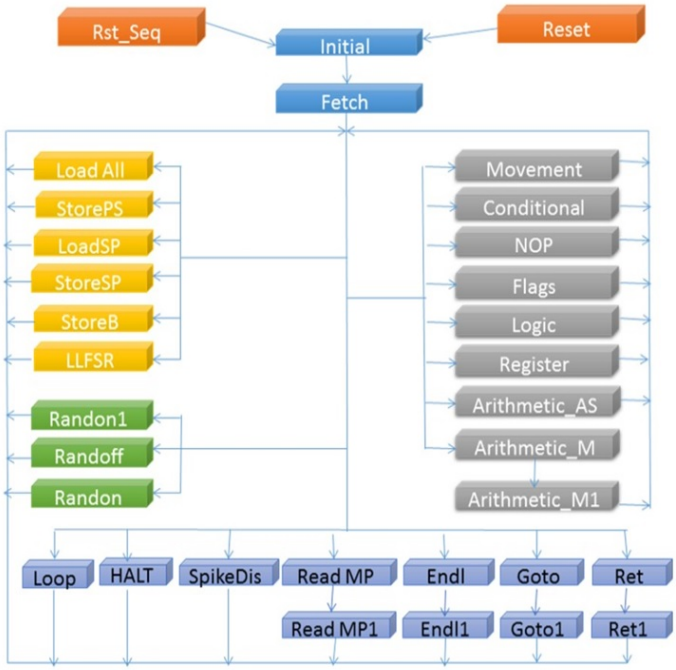


Figure 6: Sequencer state machine.

execution and initiates the off-line scan mode. In this mode the monitor buffer in each CPE is read one by one and the data is sent to the user-side communication block which in turn sends the data to the host CPU.

(b) If STOREB is followed by any other instruction other than HALT, then the sequencer initiates the On-line scan mode. This mode is the same as the off-line scan but the sequencer does not halt execution. In case the sequencer comes across another STOREB while the user-side Ethernet communication block is busy sending the Neural/Synapse information from the previous STOREB, the sequencer halts till the user-side Ethernet block is free and then continues execution. However when the sequencer is halted, entities in SNAVA can be accessed by the host CPU.

c) **Access control module** – Access control module controls host CPU access to entities in SNAVA. All the read or write access from a host CPU to the registers/BRAMs in SNAVA is through the user-side Ethernet block. The CPU access control provides access to the registers in every PE and the BRAM access switch provides access to all the synaptic and neural BRAMs. The Config unit controls and executes the access through these two blocks on request from the user-side Ethernet block.

d) **Spike generation** – This module is in charge of sending spikes that are produced during the synaptic-neural processing phase to the AER control [26] in order to carry out the spike distribution phase. The main feature of this module is to inform the AER control the details of the neurons that have fired by sending the address of the neuron to it. The length of the address is the 15 bits, which is composed by 4 bits for column, 4 bits for the row and 7 bits for the virtual neuron. These addresses are stored in a 15-bit FIFO that is included in the AER control (see Fig. 2).

a) Virtual neurons

The virtualization concept was introduced in the SNAVA architecture to improve area consumption by simulating large-scale spiking neurons. This was possible by implementing a neural BRAM, which stores the neural parameters of different neurons, in every single PE, as shown in Fig. 3. Therefore, every SNN algorithm to be implemented in SNAVA, must contain in its code the neural LOOP (LOOPN) and the synaptic LOOP (LOOPS), as shown in Fig. 7. The LOOPS and LOOPN are the two instructions that are executed by the ALU and the sequencer in order to process synaptic and neural parameters, respectively. Therefore, the synapses and virtual neurons are carried out sequentially in each PE, but all virtual neurons that are contained in the PE array are updated in parallel. As can be observed in Fig. 7, the sequencer executes LOOPS and LOOPN instructions during Phase I. The index of the Synapse loop indicates the memory position of a single synapse. Two instructions (LOADSP/STORESP) are executed inside the Synapse loop to load synaptic parameters from Synaptic

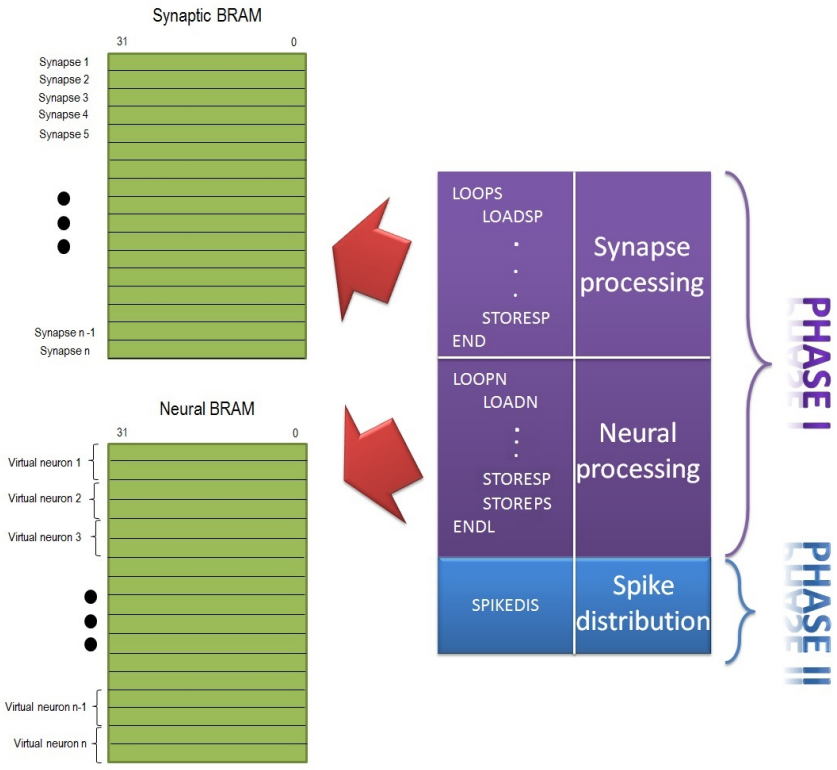


Figure 7: Multi-virtual neuron processing with assembly structure.

BRAM to active registers or viceversa (Fig. 4). The Neuron loop is used to implement the virtualization concept in PEs. If each PE is designated to simulate a single neuron, this loop is executed only once. The instruction LOADN loads neural parameters from Neural BRAM to internal active registers and instruction STOREN stores neural parameters from active registers to Neural BRAM. In this loop, the STOREPS spike instruction triggers the AER address generator to scan the PEs for Neural spikes and stores the status for every PE. It generates the addresses for the PEs that spiked at the current time and sends them to the user-side AER for transmission. In case the same PE is used to simulate several virtual neurons, there must be a STOREPS instruction at the end of every neuron loop-iteration and a SPKDIS instruction at the end of the

neuron loop. This means the AER address generator sends the addresses of the virtual neurons that spiked at the end of processing phase and the user-side AER is triggered by the sequencer only after all the virtual neurons have been processed. Here, all the PEs simulate the same number of virtual neurons and number of synapses by expending the same number of clock cycles irrespective of whether there is or not pre-synaptic spike and the spike distribution time varies depending on the resulting spikes, which are propagated into the network.

4. Implementation and performance

The SNAVA prototype has been implemented on a KC705 board from Xilinx, which includes a Xilinx Kintex-7 FPGA, to support an array of up to 100 PEs and 200 synapses per PE and a single PE can implement up to 128 virtual neurons running at 125 MHz. Hence a single SNAVA prototype could simulate up to 12,800 virtual neurons and 20,000 synapses. This FPGA board offers advanced modules of hardware that involve high speed serial links and advanced memory interfaces. The use of these has facilitated the development of a scalable architecture with high performance in terms of communication and processing. Therefore, the scalability is one of important features offers by SNAVA which allow the simulation of large-scale SNN by connecting multiple FPGAs boards. The proposed network topology is based on a ring configuration, which enables the efficient use of the resources available, namely the construction of the network does not require extra hardware such as routers to connect from 2 up to 127 FPGA boards in order to perform efficiently the distribution phase [26], as shown in Fig. 8 and the pipeline operation carried in every board increases the performance of the spike communication through the network. Hence, SNAVA is scalable respect to the number of boards and is also scalable in the number of processing elements. The user can define the number of processing elements that are required for a specific application enabling the creation of an optimized architecture.

In regards to communication, two protocols have been implemented on

SNAVA in order to manage the data flow:

- Aurora is a free communication protocol provided by Xilinx, which is used to transmit data point to point through fast serial links. This protocol offers high bandwidth, supports Full Duplex & Simplex channels and occupies minimum area [27]. Aurora has been used in SNAVA for communication of spikes between neurons supporting the synchronous AER protocol [26]. Our AER protocol uses the Ring Size parameter that defines the number of nodes in the system, and a 7-bit Chip Identifier (ChipId) to identify the source of each transmitted spike, which allows to connect up to 127 FPGA boards.

- For communication with the Host (SNAVA HMI), Ethernet MAC protocol has been selected due to its flexibility, performance and reliability. As mentioned above, the CPU access to SNAVA is by Gigabit Ethernet cables. TriMAC Ethernet IP provided by Xilinx for 7-series FPGAs has been utilized for this purpose. This IP core provides the liberty to the user in selecting between the three possible speeds of operation (10/100/1000 Mbps) [28].

As specified earlier, two Graphical User Interfaces (GUIs) have been developed in order to design, control and monitor the SNAVA network. The description of a neural model and the design of the network are done with the help of software called $SNAVA\ Conf$. $SNAVA\ Conf$ generates files required to configure the network on the prototypes as designed. These files can be seamlessly exported to SNAVA Human Machine Interface (HMI) to control the execution, monitor the network and graphically display the synaptic-neural parameters for easy perception and analysis. The current version of SNAVA HMI software can configure four FPGA boards in parallel with the designed network using the four available G-Ethernet connections (more could be configured if available). In the case of employing more than four boards, they can be initialized up to four at a time. It must be noted that only four boards can be displayed in order

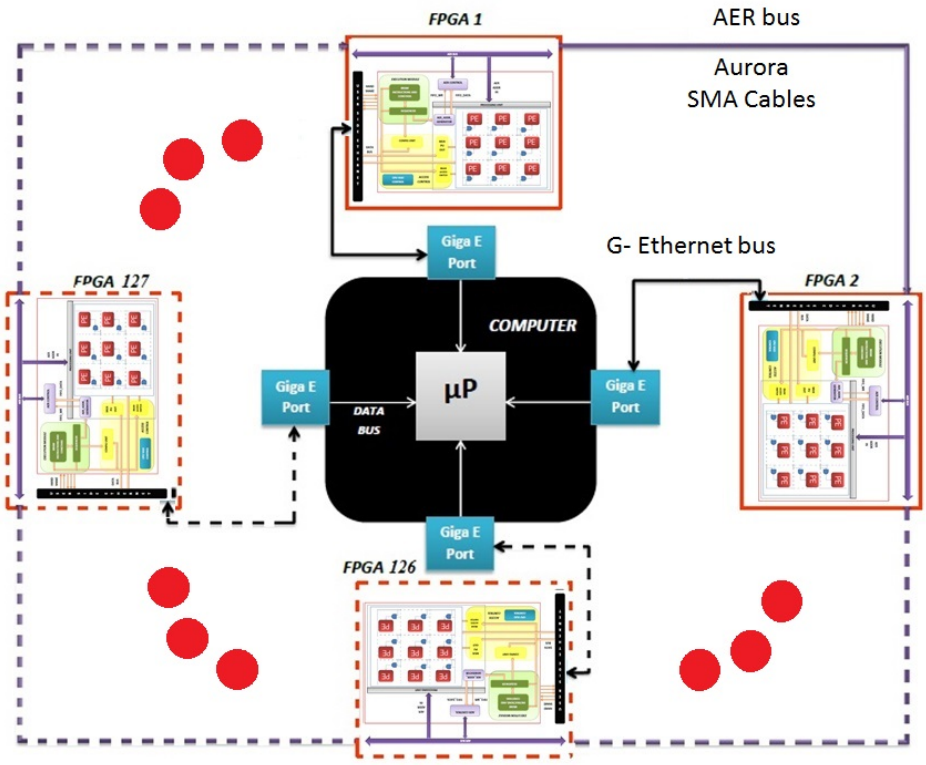


Figure 8: SNAVA communication network.

to study the neural network output. A switch can be used in order to display more boards. A brief overview of both software tools is presented below.

a) SNAVA Conf

SNAVA Conf is a specific software that supports configuration of two multiprocessor functionalities; a) configuring the neural model b) configuring the neural network. Based on these two configurations all the information required to carry out the simulation like the SNAVA compatible machine code from assembly code describing the algorithm execution, initial values of all the neural and synaptic parameters are generated as files post-processed with SNAVA HMI (see Section b). The *SNAVA Conf* main window is composed of three sections labelled as Neural model, Network, and Synapses, as shown in Fig. A1.

The Neural model section allows selecting the SNN model. There are some standard models already pre-defined in the software such as LIF, Izhikevich [9], Iglesias and Villa [25]. These existing models can be modified or even new models can be defined and added to the list of existing models. The SNN model to be implemented on the SNAVA boards is selected from the drop down box of "Model". The network can be mapped on several FPGA boards (SNAVA boards). The number of SNAVA boards to be connected to the ring network [26] is indicated in the "SNAVA Boards" text box, as shown in Fig. A1. In this arbitrary example, two boards have been enabled with 100 PEs each one and each PE contains 10 virtual neurons.

In the network section, the number of Processing Elements, virtual neurons per PE, synapses per PE and the initial synaptic-neural values of the model per SNAVA board are set. The array size is defined as rows and columns in the "PE x Row" and "PE x Col" text boxes, respectively.

The Synapses section of the main window is dedicated to allow users to graphically connect synapses between virtual neurons across all the implemented SNAVA boards. Observing Fig. A1, 'chip A' selects the source of the synaptic connection and 'Chip B' the connection destination. An arbitrary neuron can be selected by choosing the SNAVA board number and the virtual neuron position in the array. These connections are converted into a custom CAM format to configure SNAVA. This CAM text file can also be written in a excel file and loaded into $SNAVA\ Conf$ by clicking "Import File" button. Fig. A1 shows an arbitrary example, in which a single virtual neuron of Chip A can be connected to 100 virtual neurons 1 of Chip B. This connectivity was used for better viewing.

Finally, clicking on the VHD button the configuration files are generated. These files contain the SNAVA compatible machine code of the SNN algorithm for the execution module, the initial synaptic-neural parameters to be stored in the BRAMs, the initial values of the LFSRs in the PEs and the synapse connectivity table in CAM format.

b) SNAVA HMI

SNAVA HMI is an application that communicates with the SNAVA boards. It controls their execution, monitors data via Ethernet and graphically displays the user-selected network parameters, with the help of charts in real-time or offline. It is adaptable to the SNN size. It provides different kinds of chart options (e.g., waveforms and raster plot). It is possible to select the neural and synaptic parameters of interest for the graphical analysis. Two modes of analysis are possible as mentioned earlier. The static/offline analysis of the SNN lets the user display graphs using stored information belonging to previous simulations. The dynamic/real-time analysis, instead, displays the network behavior in real time by the use of dynamic charts. Here, only neural parameters such as membrane potential and post-synaptic spikes can be displayed in order to achieve 1 ms time step resolution. The main window is separated into five sections: Configuration data, Watched elements, Board communication, Graphs and Debug window as shown in Fig. A2. Firstly, on the top-left corner under Configuration data, the desired mode of analysis is selected (i.e., neural or synaptic offline or dynamic real-time analysis). For real-time analysis, all the set up files generated by $SNAVA\ Conf$ are loaded. For offline analysis, results of a previous simulation are loaded.

The text area displays the synaptic-neural parameters once these parameters were previously selected by clicking ADD button. This button will show all the neurons, synapses along with the parameters available in the network. This allows the user to select synaptic-neural parameters either choosing from the list or modifying manually the existing parameters in the text area. In the latter case, the user must previously define the format of synaptic-neural parameters (i.e., the user defines the labels according to the SNN model to be studied). The parameters specified in the text area will be used by the software during the communication phase, to filter the messages sent by SNAVA in order to collect information of only those parameters needed by the user, discarding all the others in order to save memory.

The simulation itself is initiated by pushing the Start Capture button. The Ethernet communication between SNAVA HMI and the SNAVA board is then initiated. The SNAVA board is initialized and the simulation begins. The series of buttons under graphs section are used to obtain the graphical results of the watched elements. Single or multiple waveform charts and raster plots can be generated.

5. Performance analysis

SNAVA exploits the benefits of its parallel SIMD architecture. The majority of instructions are carried out in a single clock cycle in all Processing Elements. The following evaluations were carried out to obtain the SNAVA performance figures in terms of processing speed and spike distribution time, by considering the simulation of Iglesias and Villa model [25], Izhikevich model [9] and Leaky integrate-and-fire model in 16-bit fixed point arithmetic operations. The algorithms for the simulation of the mentioned models have been programmed in assembler code in order to achieve the maximum efficiency in terms of the execution time. Also, the program was defined in a structured manner in order to simplify the process of update. The assembler codes are available in following link `http://posgrados.esimecu.ipn.mx/docs/mcimi/AnnexB.pdf`. The evaluation measures the performance of SNAVA in terms of number of clock cycles to execute one simulation cycle of the above mentioned SNN model algorithms.

The LIF model has been implemented on SNAVA in order to be used in the applications that involve processing of sensory information due to its low computational complexity. The algorithm consists of seven subroutines dedicated to compute the neural parameters and a loop to calculate synaptic parameters. The required number of cycles to execute each subroutine (neural loop and synaptic loop) in Phase 1 is indicated in Table A2.

We evaluate the Izhikevich model, which has become quite popular for the simulation of Spiking Neural Networks [9]. This is because the SNN model exhibits various spiking bursting behaviours of cortical neuron. The Izhikevich

model is claimed to be computationally as efficient as the LIF model, and also this model requires minimum hardware to simulate a large number of neurons. The required number of cycles to execute each subroutine in Phase 1 is indicated in Table A3.

The SNN model proposed by Iglesias and Villa includes the modelling of the neuron as LIF neuron but they include in the modelling the synapses important mechanisms like learning based on the Spike-Timing-Dependent-synaptic Plasticity (STDP) [25]. They also included the noise in the simulation of their model and the refractoriness in the neuron. The assembler code consists of a main loop containing nine subroutine calls which are dedicated to calculate the neural parameters, and a synapse loop which is dedicated to compute the synaptic parameters, as shown in Table A4. This synaptic loop also includes 6 additional subroutines to support plasticity mechanism in the synapses. The number of clock cycles per subroutine were measured and translated into a mathematical equation as a function of the variables S, N and N_V, where S defines the number of synapses to be simulated by the SNAVA multiprocessor, N is the number of words in the neural BRAM assigned to each virtual neuron in order to store the neural parameters and N_V is the number of virtual neurons. There is a constant number of clock cycles in each assembler code (122, 222 and 189) that do not depend on the variables mentioned above. Adding all the contributions of Tables A2, A3 and A4, equations 1, 2 and 3 allow to obtain the number of cycles N_T required to execute a single simulation cycle in SNAVA for Leaky integrate-and-fire model, Izhikevich model [9] and Iglesias and Villa model [25], respectively.

$$N_T = 4 \times N \times N_V + 122 \times N_V + 25 \times S \tag{1}$$

$$N_T = 4 \times N \times N_V + 222 \times N_V + 31 \times S \tag{2}$$

$$N_T = 4 \times N \times N_V + 189 \times N_V + 166 \times S \tag{3}$$

Monitoring the synaptic and neural parameters on real time is essential to observe the SNN dynamics. The penalty produced by this fact is not always properly reported in neural hardware simulation articles. This section measures the overhead required to relay the parameters to SNAVA HMI through user-side Ethernet. SNAVA allows the user to define the number of synaptic parameters or neural parameters to be displayed on the monitor. Neural and synaptic parameters in processing phase can be sent to SNAVA HMI once it has been processed and the required parameters are moved into Reg 0 Bank 0 within the PE. The user-side Ethernet begins transmission of the data in the monitor buffer on ad-hoc STOREB instruction. If the online scan mode is chosen, the user-side Ethernet is relaying the parameters from a previous STOREB until new STOREB is encountered or the execution module reaches the end of the processing phase, then the sequencer should stop its operation. It resumes only after the user-side Ethernet module reads the data stored in every buffer sequentially and sends these to SNAVA HMI. Equation 4 gives the number of clock cycles required by the user side module to read all buffers,

$$N_{TD} = P\left(\frac{BS}{B}\right)(S \cdot SD + N_V \cdot ND) \tag{4}$$

where N_{TD} is the number of clock cycles to display the parameters in SNAVA HMI, BS is the Ethernet interface register width (32 in the prototype), SD is the number of synapse parameters to be displayed, ND is number of neural parameters to be displayed, P is the number of PEs, and B is the Buffer size (8 in the prototype). In addition, the time required for the spike distribution needs to be taken into account. It depends on the number of virtual neurons that spike and the number of SNAVA boards in the ring. Equation 5 defines the number of clock cycles required for a single simulation cycle execution of an SNN of any model and any size.

$$N_T = K_1 \cdot N \cdot N_V + K_2 \cdot N_V + K_3 \cdot S + N_F \cdot N_{CHIPS} \tag{5}$$

Where K_1, K_2 and K_3, are SNN model dependent constants, N_V is the

number of virtual neurons per PE, S is the number of synapses per PE, N is the number of PEs, N_F is the number of neurons that spike and N_{CHIPS} is the number of SNAVA boards connected to the ring.

Theoretical figures were obtained by analysing the performance of the proposed multi-FPGA configuration involving 127 FPGAs in which all the FPGAs are updated in parallel, as shown in Fig. 8. Fig. 9 shows the execution time of a single simulation step of a single FPGA since all FPGAs are updated in parallel using three SNN models (LIF, Izhikevich [9] and Iglesias and Villa [25]) that was calculated with equations 4 and 5. The constants K_1, K_2 and K_3 were obtained from equations 1, 2 and 3, respectively. However, the spike distribution time was calculated with equation 5 taking into account the worst case (i.e., all the virtual neurons of all FPGAs fire at every simulation step). In addition, the calculation of the theoretical performance considers the time for displaying the post-synaptic spikes. Therefore, 127 FPGAs can simulate up to 1,625,600 virtual neurons with 2,540,000 synapses achieving around 1 ms by processing three large-scale SNN models with different computational complexities, as shown in Fig. 9.

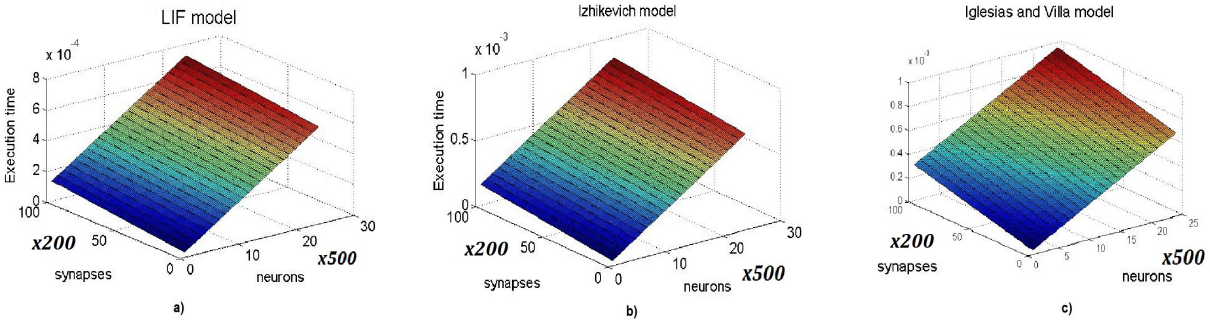


Figure 9: The execution time required to perform the LIF, Izhikevich [9] and Iglesias and Villa model [25] in a single time step

In addition to the previous analysis, a benchmark was made in terms of processing speed in order to compare SNAVA's performance with CPU-based and GPU system by using 2,000 Izhikevich neurons [9] and 10 synapses per neuron.

Device	Execution Time
This work	1.78 ms
GPU	6.96 ms
CPU	230 ms

Table 1: Execution time by performing 2,000 Izhikevich neurons [9] in three different digital devices.

These neurons were randomly connected. The Izhikevich model has been widely used as a benchmark in other architectures to verify their performance in terms of processing speed [6, 8, 10, 12, 29]. Fig. 10 shows the experimental setup to implement 2,000 Izhikevich neurons with 10 synapses per neuron in SNAVA. As it can observed, we used two boards to test the multi-board configuration. Each board contains 100 PEs and each PE supports 10 virtual neurons. The screen displays the resulting spikes of each board in their corresponding two raster plots by executing 1000 simulation steps. A single threaded neural network simulator programmed in C required 230 ms to calculate 1,000 ms of simulation time on a single thread of a 12-thread, 6-core Xeon E5-2630 2.6GHz server with 64 GB RAM. Likewise, the SNN network was programmed in C and implemented on NVIDIA GeForce GTX 890 Ti 2816 CUDA Cores 1 GHz with 6 GB GDDR5. The GPU required 6.96 ms to calculate 1,000 ms of simulation time on a single thread of 16-thread per core. SNAVA performance shows that the simulation of the SNN network required 1.78 ms to compute 1,000 ms of simulation time. Therefore SNAVA is 129 times faster than the software simulation on CPU-based and 4 times faster than GPU system, as shown in Table 1.

6. Related work

Several approaches have been proposed with the aim of providing a SNN simulator tool, which can be useful to understand some clues and structures of the brain by neuroscientists, working on real time. To achieve such purpose it

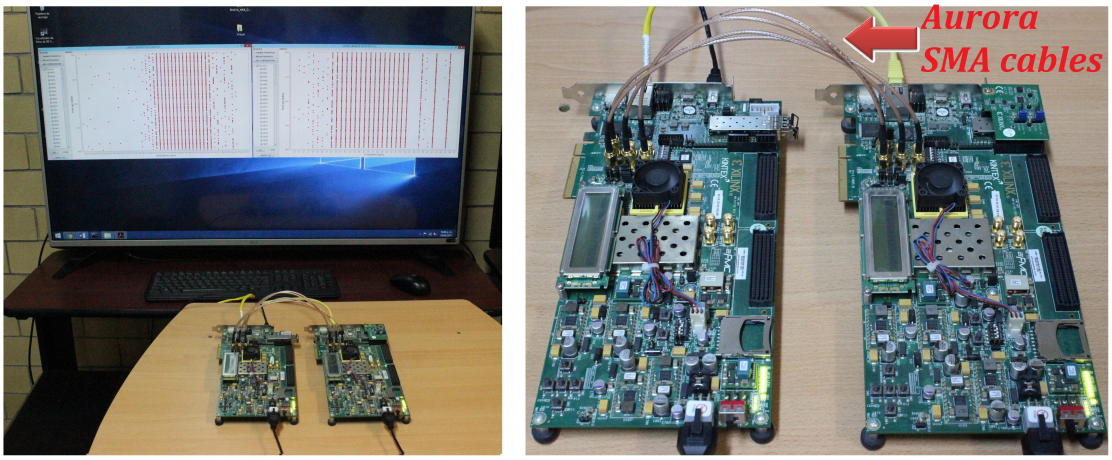


Figure 10: The overall system setup. a) Visualization of Izhikevich neurons [9] on the screen. b) Two Kintex-7 boards connected as ring configuration through SMA cables

is necessary to create a flexible architecture in terms of easy programmability for supporting different SNN models and connectivity configurations. Indeed, its architecture must be efficient in terms of processing speed and communication speed to achieve real-time (1 ms time step). In digital domain TrueNorth [5] and SpiNNaker [6] architectures have been proposed to achieve such objective. Both custom architectures promise to be powerful platforms in the simulation of large-scale of SNN models. In contrast, there has been an extraordinary development in reconfigurable computing devices within a short period of time, especially in their resource availability, speed, and configurability (FPGAs), which makes these devices suitable to simulate those networks. FPGAs provide a platform for designing highly parallel systems and high-speed serial links which are suitable for the simulation of SNN models. Obviously, these platforms are designed for general purposes making them less efficient in terms of area and power consumption in comparison with TrueNorth [5] and SpiNNaker [6] architectures (see Table 3). Clearly the implementations mentioned above can simulate many more neurons and synapses than the proof-of-concept

SNAVA. However, the strength of SNAVA is the SIMD architecture with a single programmable control unit that makes multi-model SNN simulation possible. The SIMD architecture with simple SNN tailored processing elements saves a lot of area when compared to SpiNNaker [6] that makes use of complex general purpose ARM cores. The TrueNorth implementation on the other hand is much more area efficient, i.e., it has the capacity to emulate more neurons and synapses when compared to SNAVA. But TrueNorth is an implementation customized to a single SNN model. Hence, SNAVA hits the balance between area efficiency and programmability. In addition, it is relatively inexpensive and much simpler to upgrade technology in FPGAs than in ASICs.

In the implementation of SNAVA, LUTs and registers represent the major area consumption by expending 73% and 24% of the Kintex-7 FPGA, respectively. Bigger networks can be implemented by using an advanced FPGA such as Virtex-7 and Ultrascale families. In addition to the area consumption, the Kintex-7 FPGA consumes just 0.625 W of power where 4 mW is spent by a single PE and the remaining power budget is dissipated by other modules such as access control, spike generation, sequencer, AER control and the user-side Ethernet communication block. However, SNAVA offers easy synapse and neuron implementation by simulating different SNN models at high processing speeds by paying a penalty in terms of area/power consumption. Evidently, if SNAVA is customized to support LIF neurons or Izhikevich neurons [9], the area/power consumption can be decreased and more neurons and synapses can be supported. In addition, SNAVA offers the implementation of learning mechanisms that is the current trend in advanced architectures [5, 14].

As it can be observed from Tables 2 3, current FPGA-based architectures and multicore-based architectures provide SNN simulation to be used as a tool for exploring some of the biological functions carried out in the brain through the execution of large-scale spiking neural networks. However, big challenges remain in order to create a platform which can be efficient in terms of performance, processing speed, high level of interconnectivity, support of complex neural modelling, etc.

Implementation	SNN model	Number of Neurons	Number of Synapses	Device	Area	Power	Learning
This Work	LIF ; Izhikevich [9]; Iglesias and Villa [25]; synfire chain model [30]	12,800	20,000	Kintex-7 XC7K325T	97,824 Slice registers; 148,774 Slice LUTs; 213 BRAMs; 100 DSP slices	$*$ 0.625 W	YES
Bluehive [12]	Izhikevich [9]	64,000	64,000,000	Altera Startix IV	NA	NA	NO
Minitaur [14]	LIF	65,536	16,780,000	Spartan-6 XC6SLX150	42,390 Slice registers; 21,195 Slice LUTs; 200 BRAMs	1.1 W	NO
Luo et. al. [15]	Passage of time model (POT)	101,000	100,000	Virtex-7 VC707	176,424 Slice registers; 286,455 Slice LUTs; 960 BRAMs ; 2,304 DSP48E1s	2.88 W	NO

Table 2: Feature summary of SNN implementations on compact devices and using a single FPGA.

$*$ We estimate the power consumption of the FPGA without electronics external to it using Xilinx XPower Analyzer in order to make a consistent comparison with the existing approaches.

Implementation	SNN model used	Number of Neurons	Number of Synapses	Device Used	Cores	Power consumption	Area consumption
This Work	Leaky integrate-and-fire; Izhikevich [9]; Iglesias and Villa [25]; synfire chain model [30]	12,800	20,000	Kintex-7 XC7K325T	100	625 mW	$*$ 12 mm^2 28nm process technology
TrueNorth [5]	Izhikevich [9]	1,048,567	26,543,545	Custom ASIC	4,096	< 150 mW	4.3 cm^2 28 nm process technology
SpiNNaker [6]	Izhikevich [9] ; Leaky integrate-and-fire	20,000	2,000,000	ARM9 processing	18	1 W [31]	101.64 mm^2 130 nm process technology [31]

Table 3: Feature summary of SNN implementations on compact devices and using a single chip.

$*$ We estimate the area consumption of the FPGA in mm^2 by analysing the package, pinout and die area of the 7 Series FPGA [32]. The study of area consumption takes into account the percentage of hardware resources utilization such as LUTs, registers, transceivers, DSP slices and BRAMs in order to make a consistent comparison with the existing approaches.

7. Synfire Chain Application

This section presents a synfire chain (SFC) used as a benchmark to demonstrate the SNAVA functionality. The synfire chain network, which was initially proposed by Abeles [33], is a configuration extensively studied to understand spike synchronization observed within local brain areas such as cat visual cortex, songbird premotor nucleus, and primary motor cortex [34].

A SFC shows that firing synchronization can be achieved by propagating spikes through a feed-forward synaptic topology [35] organized as several layers of excitatory neurons. Assuming a burst of spikes with a given dispersion at the input layer, the SFC imposes the condition that, at each layer of neurons the dispersion is maintained or reduced, improving the synchronization of the input excitation. On the other hand, if the dispersion between layers increases such synchronization will eventually fade.

This work presents a SFC network that is composed of 200 neurons with 50 synapses per neuron. The SFC network has 4 layers and each layer requires 50 neurons as shown in Fig. 11.

7.1. Neural Model

We use the LIF model to simulate the synfire chain model [30] for the neural activity. It is a non-linear model defined by three-variable differential equations shown as follows:

$$\begin{aligned} \tau_{mem}\frac{dv}{dt} &= x(t) - (V(t) - V_r)\,. \\ \tau_{rft}\frac{dx}{dt} &= -x(t) + y(t)\,. \\ \tau_{rft}\frac{dy}{dt} &= -y(t) + \tau_{rft}\cdot 25.27\ mV + B. \end{aligned} \tag{6}$$

where v denotes the membrane potential; x and y correspond to variables that represent the activity of ion channel currents; parameter V_r is the resting potential; τ_{mem} and τ_{rft} represent the membrane and relative refractoriness time constant, respectively. The threshold voltage V_{th} = -70mV, τ_{mem} = 10

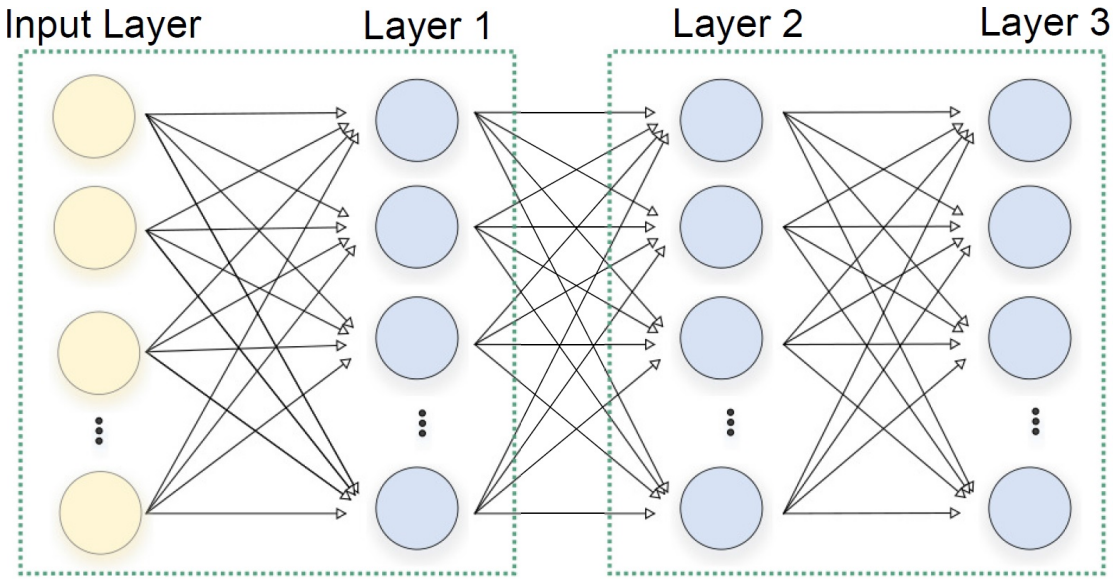


Figure 11: SFC topology

ms, $\tau_{rft} = 15$ ms and the background activity noise (B) was generated using a LFSR that is contained in each PE (see Section 3).

The synaptic model was implemented as the weighted sum of the post-synaptic spikes added to the state variable $y(t)$. We applied the same weight to all synaptic connections. This synaptic current is represented by I_s.

Equations (7) show the discrete functions implemented in SNAVA to solve the differential equations using Euler approximation, with initial conditions $V(t_0) = V_{th} - Vr$, $x(t_0) = 0$ and $y(t_0) = 0$.

$$\begin{aligned} v(t+1) &= v(t)(1 - \frac{\Delta t}{\tau_{mem}}) + (V_r + x(t))\frac{\Delta t}{\tau_{mem}} \,. \\ x(t+1) &= x(t)(1 - \frac{\Delta t}{\tau_{rft}}) + y(t)\frac{\Delta t}{\tau_{rft}} \,. \\ y(t+1) &= y(t)(1 - \frac{\Delta t}{\tau_{rft}}) + (25.27 + \frac{B}{\tau_{rft}})\Delta t + I_s \end{aligned} \tag{7}$$

7.2. Results

As mentioned above, the SFC network requires 200 neurons with 50 synapses per neuron. Therefore, a single FPGA board, which can support 100 PEs with 20,000 synapses, can be used to simulate them at high processing speeds. In this application, we use 100 PEs with 2 virtual neurons per PE and assigning 50 synapses for each virtual neuron, i.e., the LOOPN instruction is executed twice and LOOPS instruction is executed 100 times. We utilize the SNAVA HMI tool to transfer the neural model initial parameters from the PC to the FPGA through the Ethernet connection. Besides, that tool is also responsible for generating the corresponding control signals so that, once the initialization is completed the execution of the SFC application is started. Furthermore, after each SNAVA simulation cycle, the post-synaptic spike events and other neural information are sent back to the PC to be graphically represented.

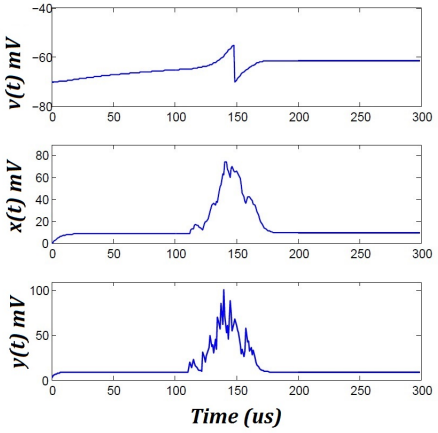


Figure 12: SFC state variables

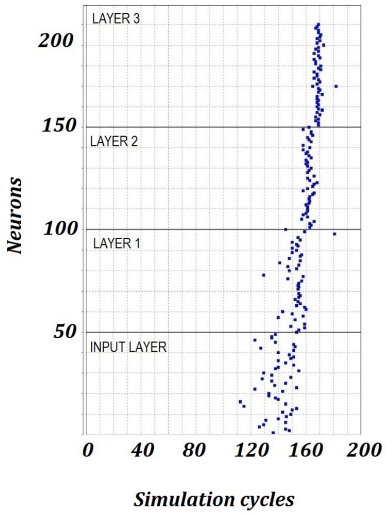


Figure 13: SFC raster plot

Since SNAVA uses 16-bit signed fixed point representation, to prevent using costly floating-point, we performed a precision analysis and determined the necessary number of decimal bits for each variable. Using 7 bits for $v(t)$ and 4 bits

for $x(t)$ and $y(t)$ a performance almost identical to the floating-point simulation is achieved [36]. Fig. 12 illustrates the state variables $v(t)$, $x(t)$ and $y(t)$ patterns activity for a single virtual neuron in the input layer. It can be seen that $y(t)$ is affected by the pre-synaptic spikes and the background activity during the spike time spread. In that way, the voltage membrane $v(t)$ is influenced by the dynamics of $x(t)$ and $y(t)$, reaching the threshold voltage and returning to the resting potential. Fig. 13 shows the SFC raster plot illustrating the spike timing of the 4 layers (200 neurons) during 200 simulation cycles. The input layer generates an input of 50 spikes with a pseudo-random distribution. It can be seen that precise timing can be sustained as the pool of input stimuli go through from one layer to another.

The preliminary results have demonstrated the functionality of SNAVA as a multi-model architecture. Our propose demonstrates more flexibility in model choice in exchange for a more simplified but less adaptable design. Other advantages of SNAVA are the availability of resources to simulate neural features such as background noise current, as well as the PE versatility to execute different algorithms. As shown in Fig. 11, neurons of the input layer are programmed to generate the input stimulus pool, while the 3 following layers process the neural algorithm. The execution time for each simulation cycle is 29.26 μs, which is inside of the biological scale rate.

8. A simple experiment involving STDP

This section presents a simple experiment that involves STDP in the synapses in order to demonstrate the capability of SNAVA for supporting such mechanism. In this experiment, we test the accuracy and precision of the calculations by stimulating a single neuron with two synapses, as shown in Fig. 14.

We model the membrane using LIF model and the synapse includes synaptic learning based on the STDP rule that was defined in Iglesias and Villa's work [25]. The membrane potential dynamics is described by the following equation:

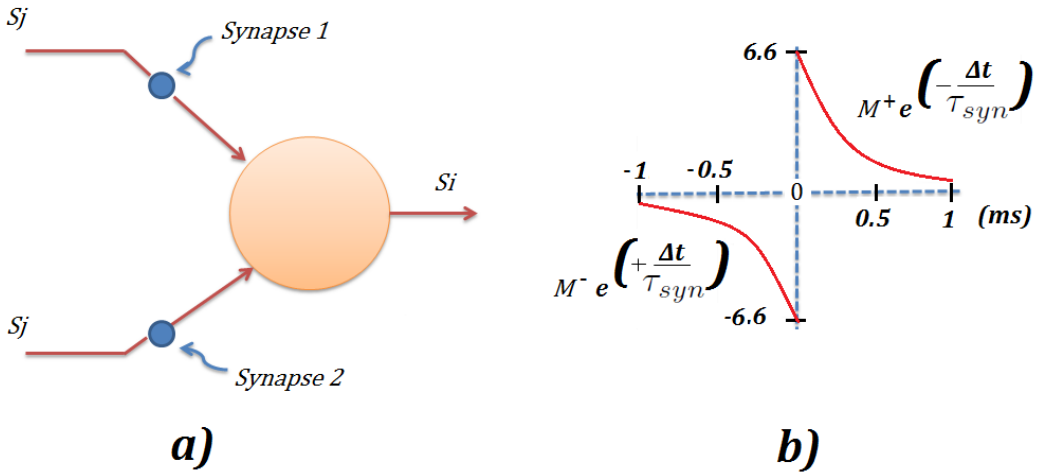


Figure 14: Diagram of a single neuron with STDP in the synapses. a) The simulation consist of a single neuron with two synapses where each of them receives an alternative stimulation for 0.6 ms. b) The STDP profile with the constants $M^+ = 6.6$, $M^- =$ -6.6, $\tau_{syn} = 0.3$ ms and $\Delta t = 0.1$ ms.

$$V(t+1) = V_{rest} + B(t) + (1 - S_i(t))\left(\left(V(t) - V_{rest}\right)k_{mem}\right) + \sum_j \omega_{ji}(t) \qquad (8)$$

where $V(t+1)$ refers to the membrane potential of neuron type either excitatory or inhibitory, V_{rest} defines the value of the resting potential, $B(t)$ is the background activity noise, $S_i(t)$ is the post-synaptic spike, $k_{mem} = e^{\left(-\frac{\Delta t}{\tau_{mem}}\right)}$ is the time constant linked to the leakage current, and finally, $\omega_{ji}(t)$ is the pre-synaptic potential. In this experiment, the resting potential V_{rest} = - 78 mV, $\tau_{mem} = 10$ ms and $\Delta t = 0.1$ ms. Here, the background activity noise $B(t)$ is an injected random excitatory pre-synaptic potential with zero mean and 30 mV standard deviation that produces an increment of the membrane potential $V(t+1)$. The neuron fires post-synaptic spikes due to noise at an average rate of 2 spikes/s.

The pre-synaptic potential $\omega_{ji}(t)$ is a function of the pre-synaptic spike S_j, the synaptic potential $P = 1$ mV, and the variable learning L_{ji} (the initial value of $L_{ji} = 20$). This is expressed by the following equation:

$$\omega_{ji}(t+1) = S_j \cdot L_{ji}(t) \cdot P \tag{9}$$

Here, the value of L_{ji} at time t defines the STDP rule according to Fig. 14b). Hence, L_{ji} is a function that depends on the correlation between the arrival of pre-synaptic synaptic spikes S_j and the generation of post-synaptic spikes $S_i(t)$. The value L_{ji} increases when there is a post-synaptic spike ($S_i(t) = 1$). This increment corresponds to the decreasing function of the elapsed time from the previous pre-synaptic spike $S_j(t)$. In a similar way, the variable L_{ji} decreases by a function of the elapsed time from the previous post-synaptic spike $S_i(t)$ and the arrival of pre-synaptic spike ($S_j(t) = 1$). This rule can be defined as follows:

$$L_{ji}(t+1) = L_{ji}(t) + (S_i(t) \cdot M_j(t))) + (S_j(t) \cdot M_i(t))) \tag{10}$$

where $M_j(t)$ and $M_i(t)$ correspond to the increasing and decreasing functions of the variable L_{ji}. These variables are defined as inter-spike decay functions. Therefore, $M_j(t)$ and $M_i(t)$ indicate the latest pre-synaptic and post-synaptic spikes, respectively. These functions are described as follows:

$$M_i(t+1) = S_i(t) \cdot M^+ + (1 - S_i(t)) \cdot M_i(t) \cdot k_{syn} \tag{11}$$

$$M_j(t+1) = S_j(t) \cdot M^- + (1 - S_j(t)) \cdot M_j(t) \cdot k_{syn} \tag{12}$$

The value of M_i and M_j are set to their maximum value M^- and M^+, respectively, when a post-synaptic spike $S_i(t)$ and a pre-synaptic spike $S_j(t)$ are generated. Otherwise, the variables M_i and M_j decay with a time constant $k_{syn} = e^{\left(-\frac{\Delta t}{\tau_{syn}}\right)}$, where τ_{syn} is the synaptic plasticity time. In this experiment, the variables M_i and M_j are set to -6.6 and 6.6 when there are a post-synaptic spike $S_i(t)$ and a pre-synaptic spike $S_j(t)$, respectively, and both decay at τ_{syn} = 0.3 ms and $\Delta t = 0.1$ ms.

8.1. Results

Figure 15 shows the results of the simulation of a single neuron with STDP in the synapses. The neuron is stimulated externally by the background activity to fire post-synaptic spikes S_i at specific times that appear in the top line. In addition, the neuron is externally fed with two pre-synaptic spikes S_j through its synapse 1 and synapse 2, respectively. In the present study, the whole simulation last 3 ms where each simulation step is equal to 1.5 μs, i.e., SNAVA computes the LIF model during 2,000 simulation steps. The first pre-synaptic spike S_j is sent to synapse 1 at 0.2 ms which causes the value of M_j is set to 6.66 and the value of L_{ji} of synapse 1 increases. However, the value of L_{ji} of synapse 1 is decreased in function of variable M_i when the neuron fires the first post-synaptic spike S_i. The first pre-synaptic spike S_j is fed to neuron through its synapse 2 after 0.6 ms producing an increment of variable L_{ji} of synapse 2 but the post-synaptic spike S_i produces a small decrement of this variable. Because of the timing between pre- and post-synaptic spikes, synapse 1 is getting depressed while synapse 2 is getting potentiated. Pre-synaptic spike 1 is distant in time from the next post-synaptic spike and in fact it is closer to the previous post-synaptic spike S_i. This produces the synapse depression. On the contrary, pre-synaptic spike 2 is close to next post-synaptic spike S_i and distant to the previous one, so it gets potentiated.

9. Conclusions

A scalable, program flexible SNN architecture has been introduced for efficient SNN simulation. Scalability is one of the main features offered by SNAVA which allows the simulation of large-scale SNN by connecting multiple FPGAs boards. The proposed network topology is based on a ring configuration which enables the efficient use of the available resources, namely the construction of the network does not require extra hardware to connect a large number of FPGA boards. This reduces the effort for making expensive dedicated interfaces to connect large number of boards like [12]. Besides, the pipeline operation carried

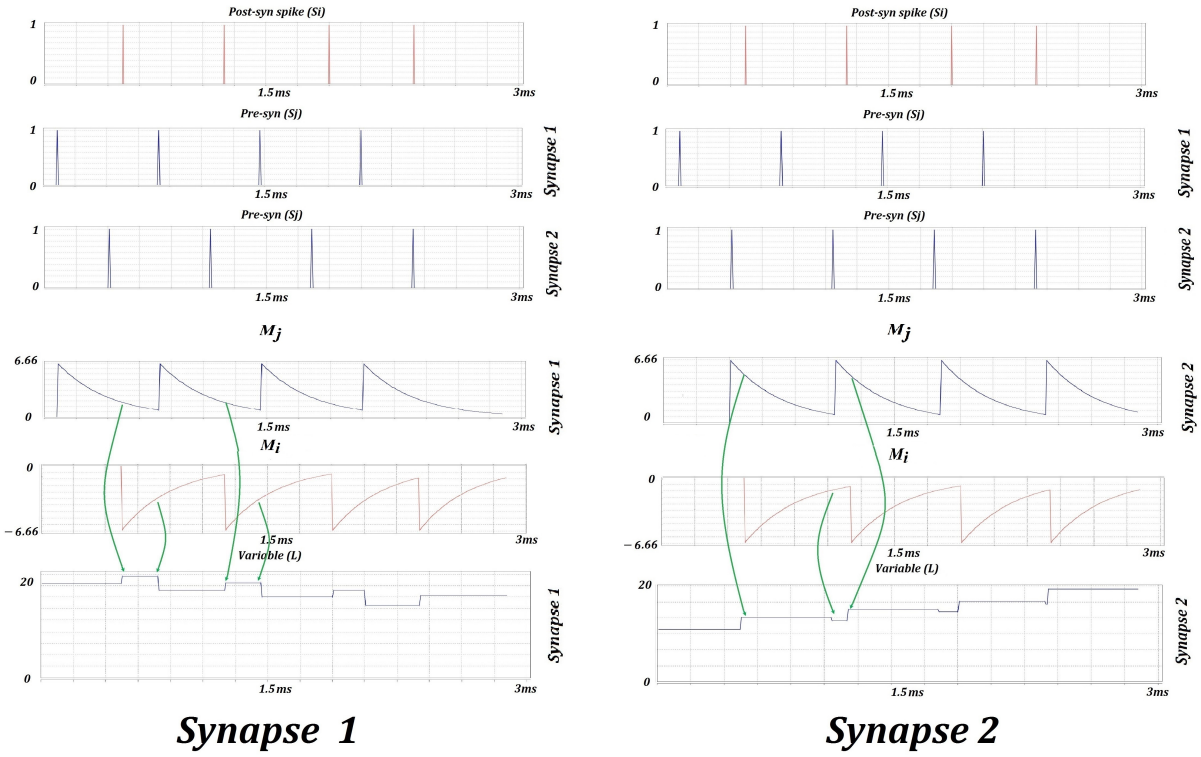


Figure 15: Graphical representation of the evolution of the strength between S_i(t), S_j(t), M_i(t) and M_j(t) variables for the synapses 1 and 2.

in every board increases the performance of the communication through the network. Here, specific time slots are used for sending spikes via the shared channel to obtain the minimum error rate (free collision transmission). In this way the spikes contention is avoided. The program flexibility is shown by the simulation of three SNN models with different levels of computational complexity. Therefore, this bio-inspired architecture offers an interesting development tool to support different SNN models by exploiting the FPGA re-configurability and minimizing implementation time.

Several ideas were applied to design an efficient architecture in terms of processing speed. There are three important contributions to build a high performance simulator. The first contribution is focused to the development of

simple, special-purpose PEs and customized instructions in order to increase the performance of the processing system. The second contribution is linked to the use of distributed memory system; the proposal idea was to implement the synaptic and neural parameters on BRAMs. The CPE accesses directly to the BRAMs by expending a single clock cycle. The third contribution is related to the implementation of 3-stage pipelined sequencer that allows to increase the processing speed. SNAVA provides distinctive features in order to implement specific SNN model for different applications: multi-model support, connection between neurons is point to multi-point, virtual neuron support. Even though SNAVA trades off some area to offer all these features, it would make fast prototyping of multi-model SNN solutions requiring real-time processing speed possible.

We developed a bio-inspired application that involves Synfire chains to demonstrate the easy programmability and high performance of SNAVA. These neural networks, which are used to analyse sequences of neural spikes, could be located in some brain areas (primary motor cortex, visual cortex). Hence, these sequences could serve as an method for learning temporal tasks such as perceptual discriminations of temporal signals and well-timed motor actions [34].

Part of future work is to improve the current GUIs in order to automatize the mapping of very large-scale SNNs. In addition, we need to develop new hardware strategies to implement variable dendritic and axonal delays and optimize the area consumption of SNAVA by implementing a larger number of synapses and neurons.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Innovation and the European Social Fund (ESF) under Projects TEC2011-27047 and TEC2015-67278-R. Giovanny Sanchez holded research fellowships supported by the Catalan Department of Innovation, Universities and Companies, and the ESF, and the Consejo Nacional de Ciencia y Tecnologia (CONA-

CyT).

References

[1] R. Ananthanarayanan, S. K. Esser, H. D. Simon, D. S. Modha, The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses, in: High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on, IEEE, 2009, pp. 1–12.

[2] T. Sharp, F. Galluppi, A. Rast, S. Furber, Power-efficient simulation of detailed cortical microcircuits on SpiNNaker, Journal of Neuroscience Methods 210 (1) (2012) 110–118.

[3] J. Schemmel, D. Bruderle, A. Grubl, M. Hock, K. Meier, S. Millner, A wafer-scale neuromorphic hardware system for large-scale neural modeling, in: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, IEEE, 2010, pp. 1947–1950.

[4] R. Brette, W. Gerstner, Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity, J. Neurophysiol. 94 (2005) 3637 – 3642, article.

[5] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, D. S. Modha, A million spiking-neuron integrated circuit with a scalable communication network and interface, Science 345 (6197) (2014) 668–673.

[6] Xin Jin, M. Lujan, L. A. Plana, S. Davies, S. Temple, S. B. Furber, Modeling Spiking Neural Networks on SpiNNaker, Computing in Science & Engineering 12 (5) (2010) 91–97.

[7] H. E. Plesser, M. Diesmann, M.-O. Gewaltig, A. Morrison, NEST: The Neural Simulation Tool, in: D. Jaeger, R. Jung (Eds.), Encyclopedia of

Computational Neuroscience, Springer New York, New York, NY, 2013, pp. 1–4.

[8] R. Hoang, D. Tanna, L. Jayet Bray, S. Dascalu, F. Harris, A novel cpu/gpu simulation environment for large-scale biologically realistic neural modeling, Frontiers in Neuroinformatics 7 (2013) 19.

[9] E. Izhikevich, Simple model of spiking neurons, IEEE Transactions on Neural Networks 14 (6) (2003) 1569–1572.

[10] A. K. Fidjeland, E. B. Roesch, M. P. Shanahan, W. Luk, NeMo: a platform for neural modelling of spiking neurons using GPUs, in: Application-Specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on, IEEE, 2009, pp. 137–144.

[11] Y. Kuramoto, Chemical Oscillations, Waves, and Turbulence, Vol. 19 of Springer Series in Synergetics, Springer Berlin Heidelberg, Berlin, Heidelberg, 1984.

[12] S. W. Moore, P. J. Fox, S. J. Marsh, A. Mujumdar, others, Bluehive-a field-programable custom computing machine for extreme-scale real-time neural network simulation, in: Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on, IEEE, 2012, pp. 133–140.

[13] C. Zamarreno-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, B. Linares-Barranco, Multicasting mesh aer: A scalable assembly approach for reconfigurable neuromorphic structured aer systems. application to convnets, IEEE Transactions on Biomedical Circuits and Systems 7 (1) (2013) 82–102.

[14] D. Neil, S. C. Liu, Minitaur, an event-driven fpga-based spiking network accelerator, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22 (12) (2014) 2621–2628.

[15] J. Luo, G. Coapes, T. Mak, T. Yamazaki, C. Tin, P. Degenaar, Real-time simulation of passage-of-time encoding in cerebellum using a scalable fpga-based system, IEEE Transactions on Biomedical Circuits and Systems 10 (3) (2016) 742–753.

[16] S. Song, K. D. Miller, L. F. Abbott, Competitive Hebbian learning through spike-timing-dependent synaptic plasticity, Nat Neurosci 3 (9) (2000) 919–926.

[17] N. Caporale, Y. Dan, Spike Timing–Dependent Plasticity: A Hebbian Learning Rule, Annual Review of Neuroscience 31 (1) (2008) 25–46.

[18] A. Carnell, An analysis of the use of hebbian and anti-hebbian spike time dependent plasticity learning functions within the context of recurrent spiking neural networks, Neurocomputing 72 (4–6) (2009) 685 – 692, brain Inspired Cognitive Systems (BICS 2006) / Interplay Between Natural and Artificial Computation (IWINAC 2007).

[19] F. Walter, F. Röhrbein, A. Knoll, Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks, Neurobiologically Inspired Robotics: Enhanced Autonomy through Neuromorphic Cognition 72 (2015) 152–167.

[20] A. Smith, L. McDaid, S. Hall, A compact spike-timing-dependent-plasticity circuit for floating gate weight implementation, Neurocomputing 124 (2014) 210 – 217.

[21] D.-H. Kang, H.-G. Jun, K.-C. Ryoo, H. Jeong, H. Sohn, Emulation of spike-timing dependent plasticity in nano-scale phase change memory, Neurocomputing 155 (2015) 153 – 158.

[22] A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodriguez, A. Jiménez, M. Rivas, G. Jiménez, A. Civit, On the aer convolution processors for fpga, in: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 4237–4240.

[23] A. L. Hodgkin, A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, The Journal of physiology 117 (4) (1952) 500–544.

[24] F. Gabbiani, C. Koch, Coding of Time-Varying Signals in Spike Trains of Integrate-and-Fire Neurons with Random Threshold, Neural Computation 8 (1) (1996) 44–66.

[25] J. Iglesias, J. Eriksson, F. Grize, M. Tomassini, A. E. Villa, Dynamics of pruning in simulated large-scale spiking neural networks, Biosystems 79 (1-3) (2005) 11–20.

[26] T. Dorta, M. Zapata, J. Madrenas, G. Sánchez, AER-SRT: Scalable spike distribution by means of synchronous serial ring topology address event representation, Neurocomputing 171 (2016) 1684–1690.

[27] LogiCORE IP Aurora 8B/10B v5.3 User Guide (UG353).
URL `https://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b_ug353.pdf`

[28] LogiCORE IP Tri-Mode Ethernet MAC v5.5 (PG051).
URL `https://www.xilinx.com/support/documentation/ip_documentation/tri_mode_eth_mac/v5_5/pg051-tri-mode-eth-mac.pdf`

[29] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, A. V. Veidenbaum, A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors, Neural Networks 22 (5-6) (2009) 791–800.

[30] M. Diesmann, M. Gewaltig, A. Aertsen, Stable propagation of synchronous spiking in cortical neural networks, Nature 402 (1999) 599–533.

[31] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, S. B. Furber, Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation, IEEE Journal of

Solid-State Circuits 48 (8) (2013) 1943–1953. doi:10.1109/JSSC.2013.2259038.

[32] 7 Series FPGAs Packaging and Pinout v1.14 (UG475).
URL https://www.xilinx.com/support/documentation/user_guides/ug475_7Series_Pkg_Pinout.pdf

[33] M. Abeles, Corticonics: Neuronal Circuits of the Cerebral Cortex, 1st Edition, Cambridge University Press, Cambridge, England, 1991.

[34] J. K. Jun, D. Z. Jin, Development of neural circuitry for precise temporal sequences through spontaneous activity, axon remodeling, and synaptic plasticity, PLOS ONE 2 (8) (2007) 1–17.

[35] H. Brama, S. Guberman, M. Abeles, E. Stern, I. Kanter, Synchronization among neuronal pools without common inputs: in vivo study, Brain Structure and Function 220 (6) (2015) 3721–3731.

[36] M. Zapata, J. Madrenas, Synfire chain emulation by means of flexible snn modeling on a simd multicore architecture, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8681 (September) (2014) 222–229.

Appendix A: Figures and Tables of SNAVA

This Appendix presents the Figures and Tables that specify the operation of SNAVA.

Instruction Category (keyword)	Function
CPE Instructions	
Arithmethic	Addition, Subtraction, saturated and unsaturated Multiplication, Increment, Decrement, Negation
Logical	And, Or, Xor

Register	Set, reset and Shift operation on CPE registers
Flags	Set and Reset of Zero, Carry, Saturation Flags
Conditional(FREEZE)	Freezes the CPE operation based on Zero, Carry, Saturation Flags
Movement	Copies data from one register to other or between Reg Banks in CPE
LFSR	Starts, stops the Pseudo random number generation in CPEs
Load Instructions	
Load all /$LFSR$	Loads data from Sequencer to the CPE register/ LFSR register
Load Synapse (LOADSP)	Loads synapse parameters from Synapse BRAM to CPE registers
Load Neuron (LOADN)	Loads Neuron parameters from Neuronal BRAM to CPE registers
Store Instructions	
Store Synapse (STORESP)	Stores the synapse parameters from the CPE registers back to the Synapse BRAM
Store Neuron (STOREN)	Stores the Neuron parameters from the CPE registers back to the Neural BRAM
Store Pre-Synaptic Spike (STOREPS)	Stores the Spike bit of CPE to AER Address Gen
Store Buffer (STOREB)	Stores data from CPE registers to a buffer which is to be sent to Host PC using Ethernet Protocol
Control and Looping Instructions	
Loop (LOOPS, LOOPN), End loop	Neuron loop and Synapse loop
Goto (GOTO, GOTONL), Return	Conditional and Unconditional Jump

Halt	Halt SNAVA
Spike distribution (SPKDIS)	Starts spike distribution phase

Table A1: SNAVA SET OF INSTRUCTIONS

Symbol	Subroutine	Clock Cycles
LNP	Load Neural Parameters	$2 \cdot N^*$
SNP	Save Neural Parameters	$2 \cdot N^*$
MP	Membrane Potential	39
CS	Cycles per synapse	$(25) \cdot S^*$
SU	Spike update	48
RF	Refractory Period	5
NS	Neuron display	24
SE	Spikes enable	6

(a) NEURAL LOOP

Symbol	Subroutine	Clock Cycles
SL	Synapse Load	3
SW	Synaptic weight	21
SS	Synapse Save	1

(b) SYNAPTIC LOOP

Table A2: a), b) Subroutine calls for executing LIF model.
N^* is the number of words in the neural BRAM assigned to each virtual neuron in order to store the neural parameters.
S^* is the number of synapses.

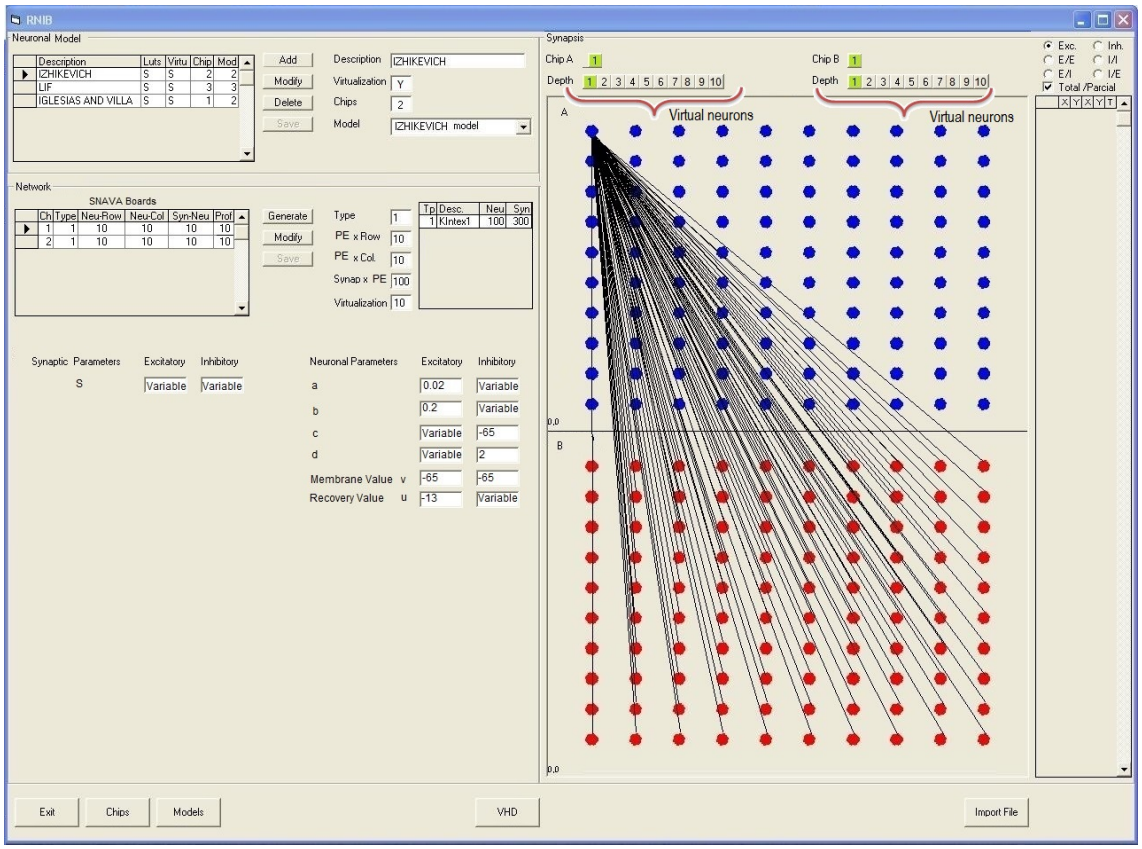


Figure A1: *SNAVA Conf* main window.

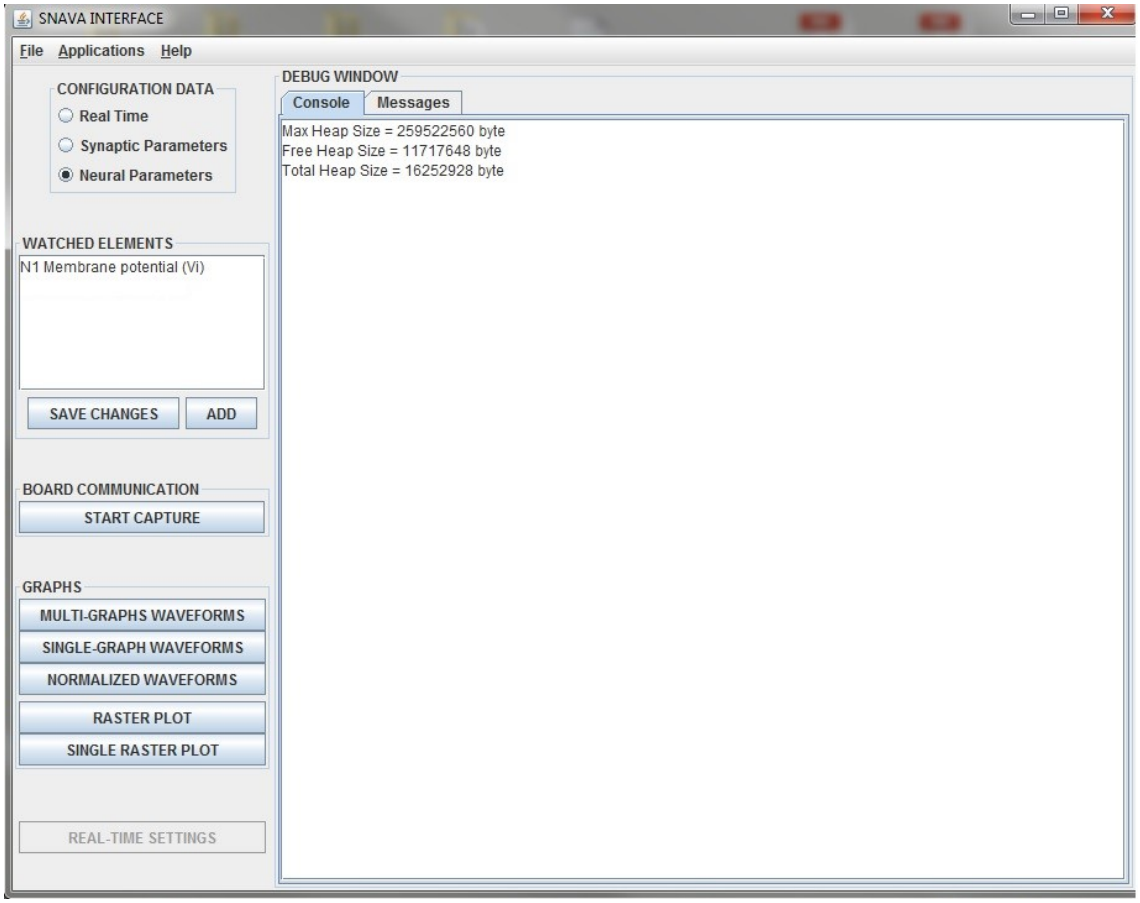


Figure A2: SNAVA HMI main window.

Symbol	Subroutine	Clock Cycles
LNP	Load Neural Parameters	$2 \cdot N^*$
SNP	Save Neural Parameters	$2 \cdot N^*$
TI	Thalamic Input	36
SU	Spike Update	48
MP	Membrane Potential	24
SE	Spike Enable	6
CS	Cycle per synapse	$(31) \cdot S^*$
MV	Membrane Value	84
RV	Recovery Value	24

(a) NEURAL LOOP

Symbol	Subroutine	Clock Cycles
SL	Synapse Load	1
SW	Synaptic weight	25
SS	Synapse Save	5

(b) SYNAPTIC LOOP

Table A3: a), b) Subroutine calls for executing Izhikevich model [9].
N^* is the number of words in the neural BRAM assigned to each virtual neuron in order to store the neural parameters.
S^* is the number of synapses.

Symbol	Subroutine	Clock Cycles
LNP	Load Neural Parameters	$2 \cdot N^*$
SNP	Save Neural Parameters	$2 \cdot N^*$
IC	Initial conditions	10
MV	Membrane value	43
CS	Cycle per synapse	$(166) \cdot S^*$
MOLP	Memory of last post-synaptic	28
SU	Spike Update	63
BA	Background activity	34
RP	Refractory period	5
SE	Spike enable	6

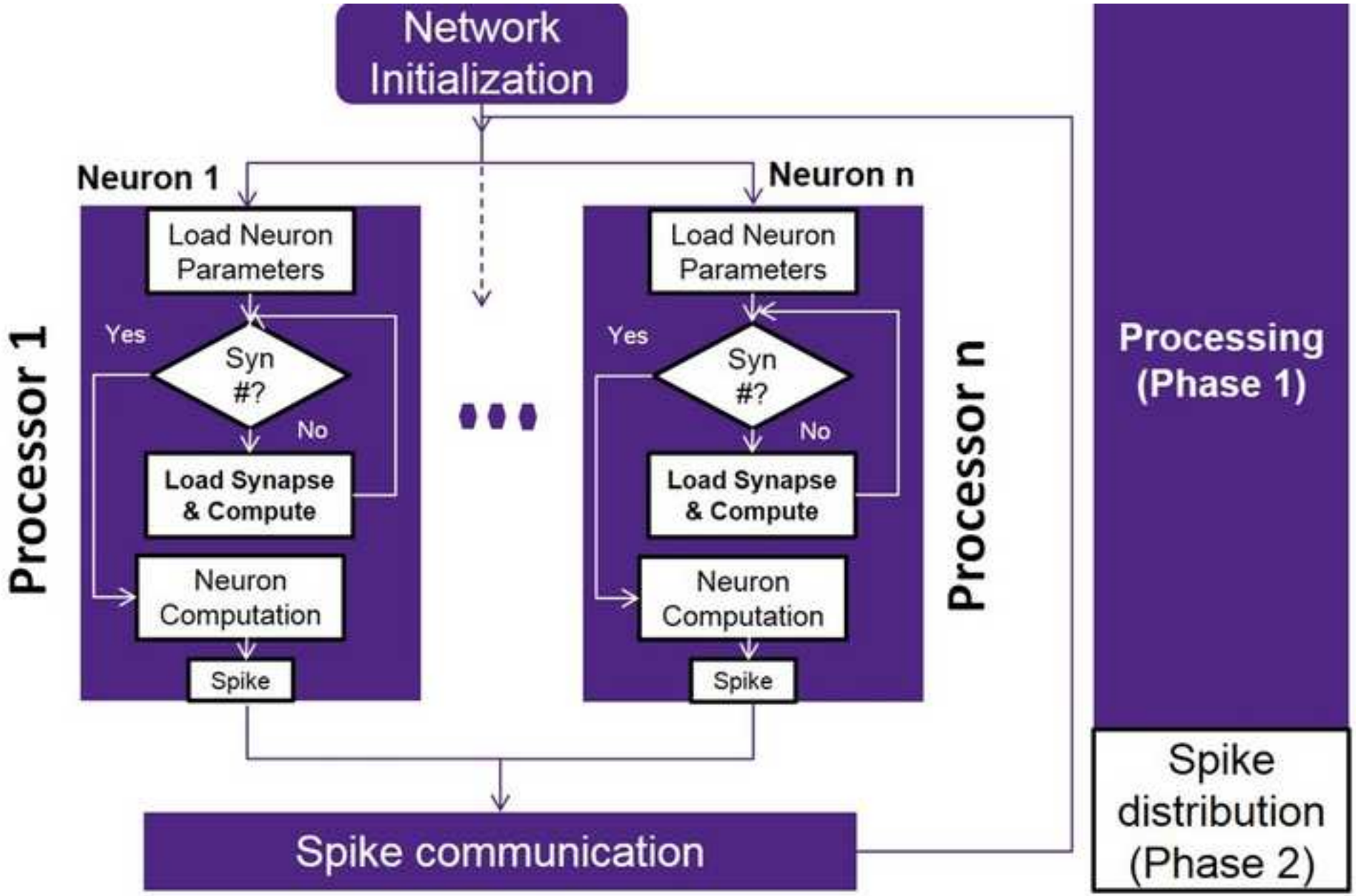
(a) NEURAL LOOP

Symbol	Subroutine	Clock Cycles
SL	Synapse Load	3
SW	Synaptic weight	37
RVV	Real value variable	30
AV	Activation variable	57
MOLP	Memory of last pre-synaptic spike	34
SS	Synapse Save	5

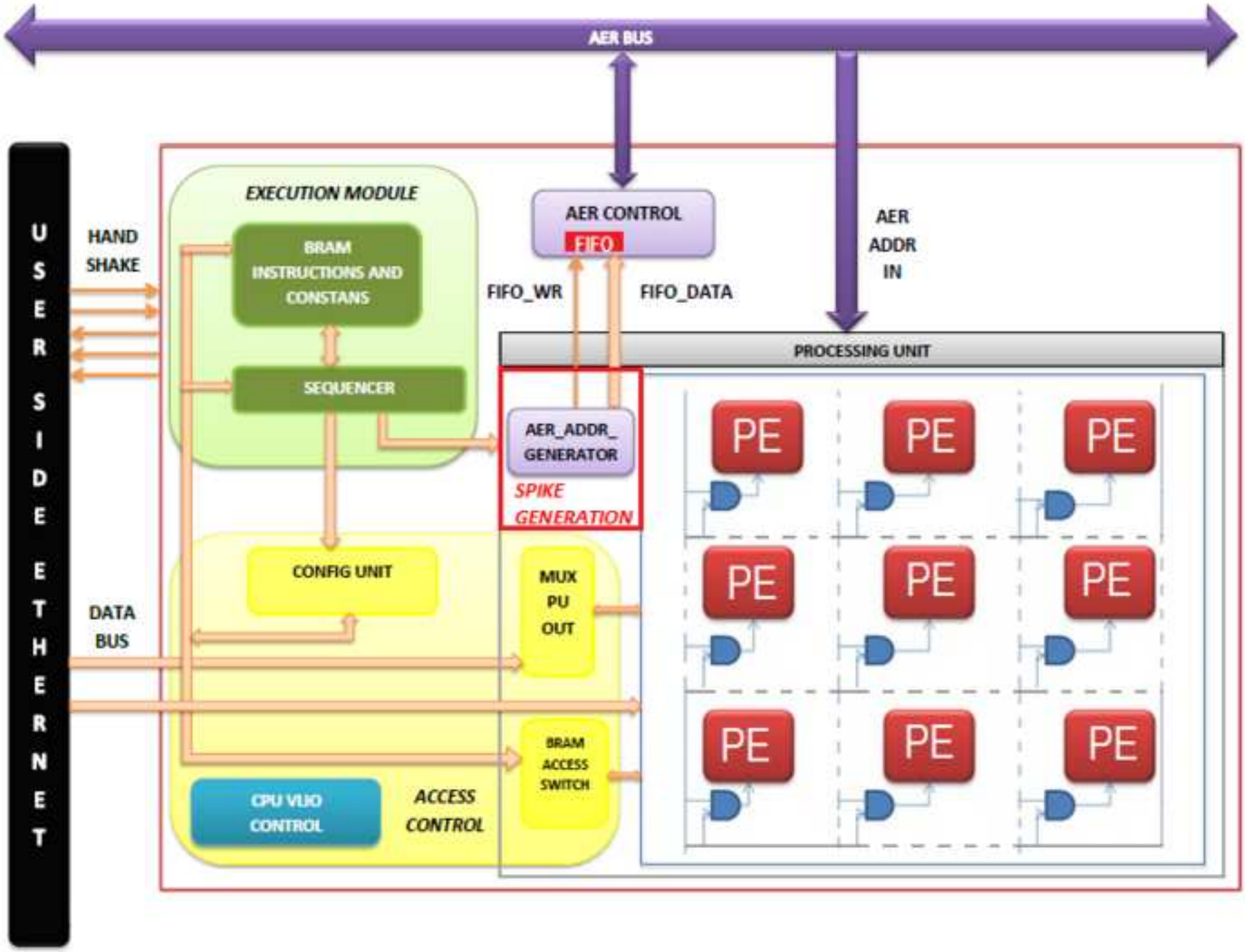
(b) SYNAPTIC LOOP

Table A4: a), b) Subroutine calls for executing Iglesias and Villa model [25].
N^* is the number of words in the neural BRAM assigned to each virtual neuron in order to store the neural parameters.
S^* is the number of synapses.

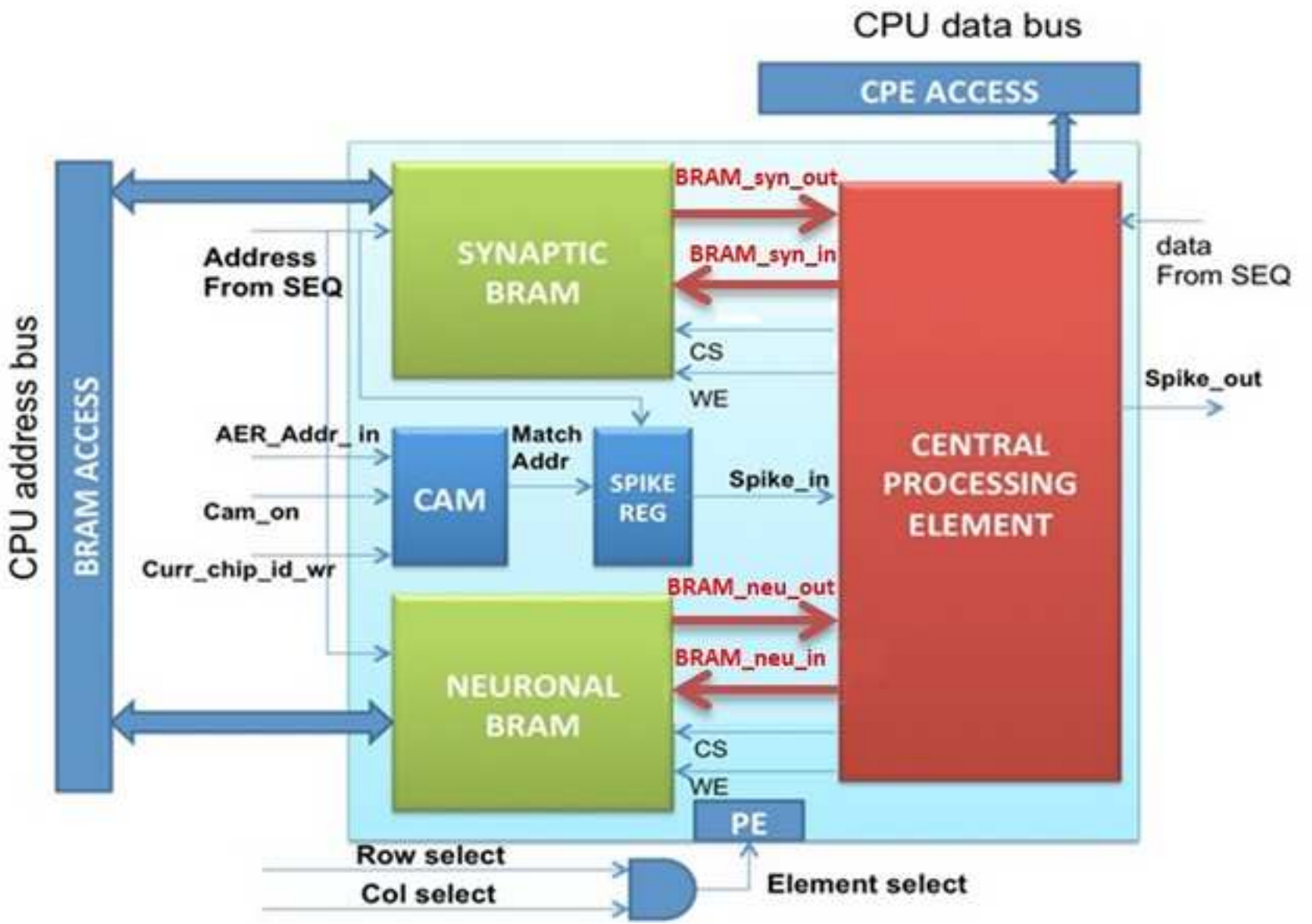
Figure


Network Initialization
Neuron 1
Neuron n
Load Neuron Parameters
Load Neuron Parameters
Yes
Syn #?
No
Yes
Syn #?
No
Load Synapse & Compute
Load Synapse & Compute
Neuron Computation
Neuron Computation
Spike
Spike
Processor 1
Processor n
Processing (Phase 1)
Spike communication
Spike distribution (Phase 2)

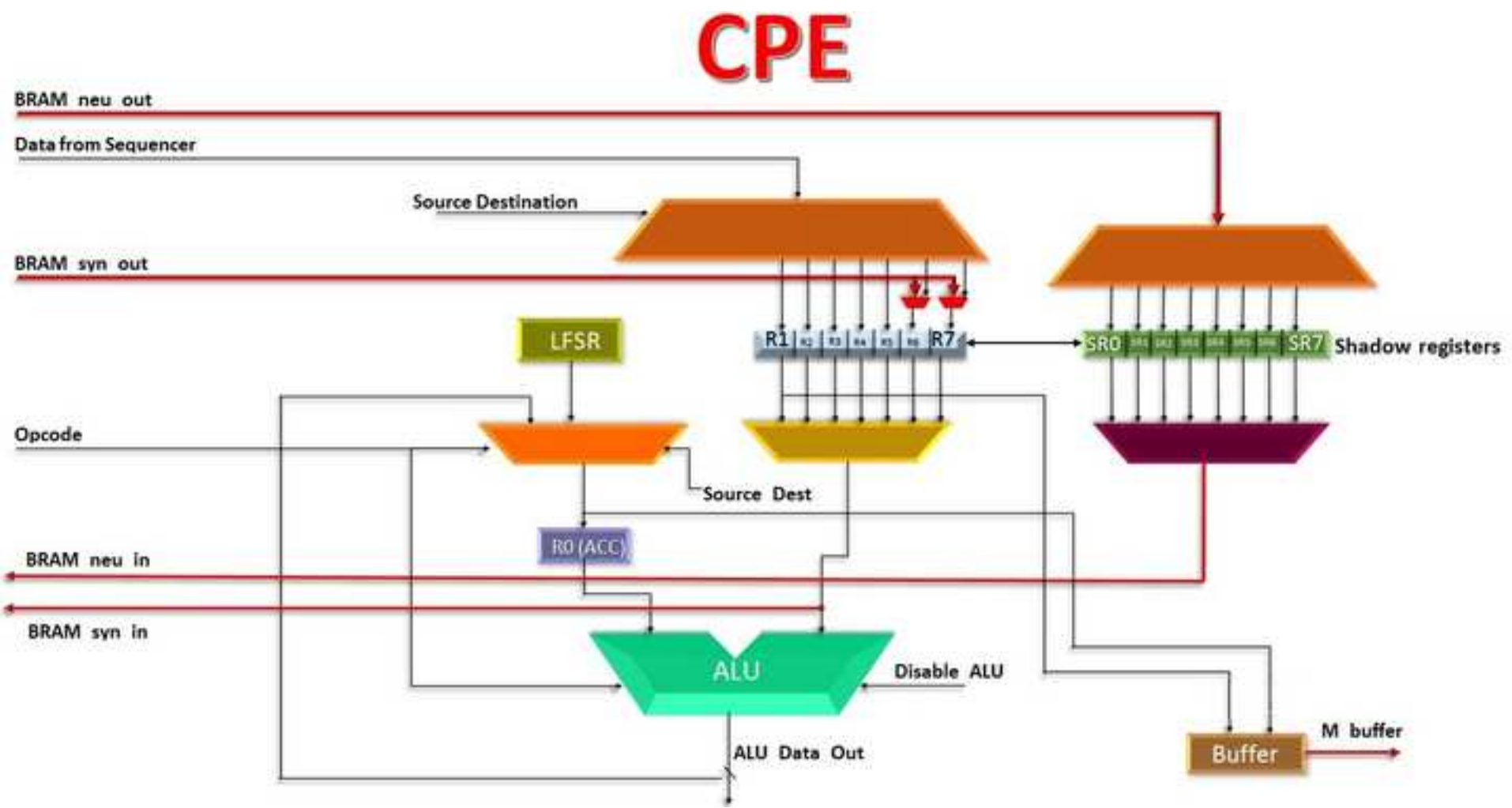
Figure


AER BUS
USER SIDE ETHERNET
HAND SHAKE
DATA BUS
EXECUTION MODULE
BRAM INSTRUCTIONS AND CONSTANS
SEQUENCER
CONFIG UNIT
CPU VLIO CONTROL
ACCESS CONTROL
AER CONTROL
FIFO
FIFO_WR
FIFO_DATA
AER ADDR IN
PROCESSING UNIT
AER_ADDR_ GENERATOR
SPIKE GENERATION
MUX PU OUT
BRAM ACCESS SWITCH
PE

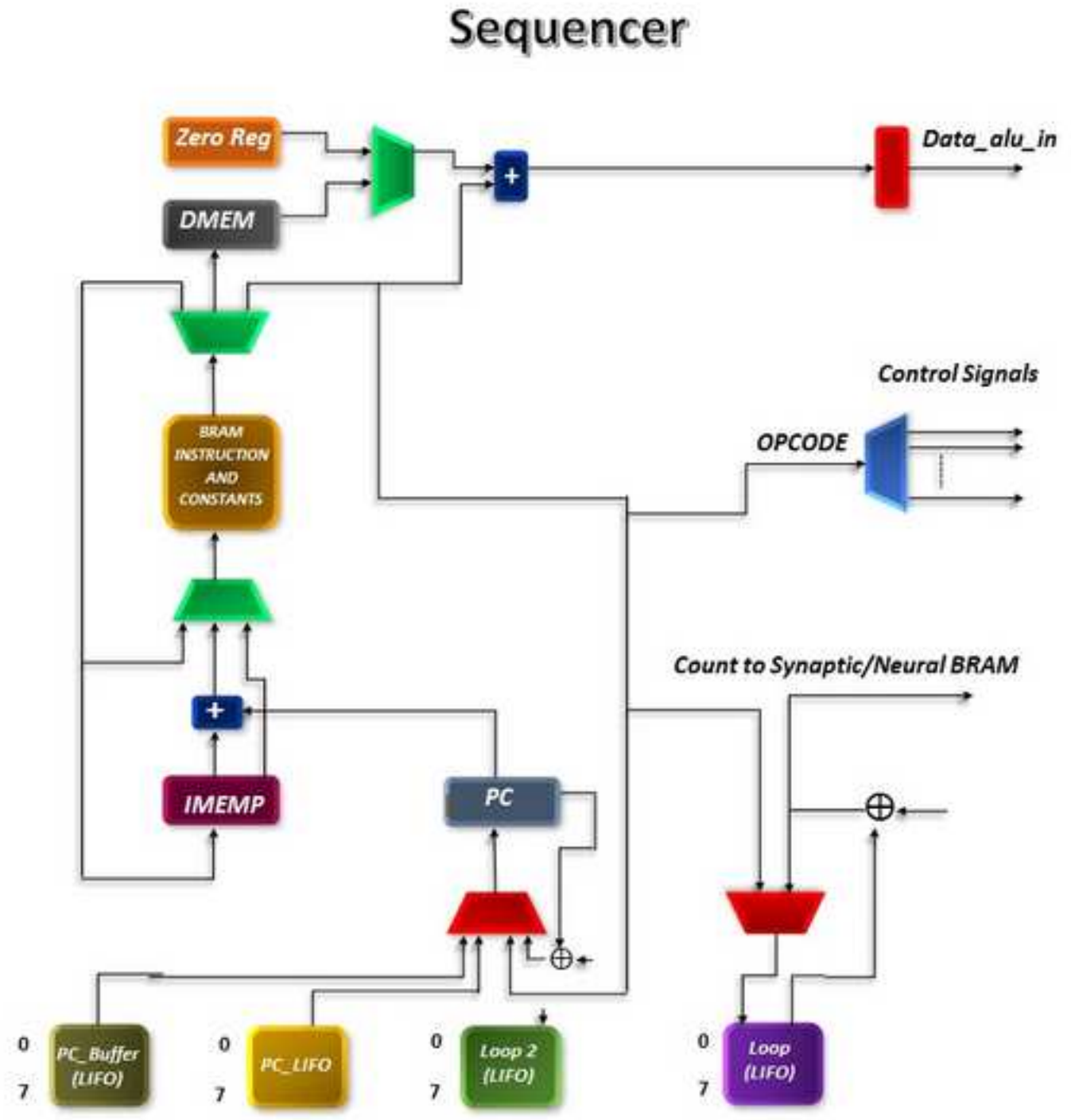
Figure


CPU data bus
CPE ACCESS
CPU address bus
BRAM ACCESS
Address
From SEQ
SYNAPTIC
BRAM
BRAM_syn_out
BRAM_syn_in
CS
WE
data
From SEQ
Spike_out
AER_Addr_ in
Cam_on
Curr_chip_id_wr
CAM
Match
Addr
SPIKE
REG
Spike_in
CENTRAL
PROCESSING
ELEMENT
BRAM_neu_out
BRAM_neu_in
NEURONAL
BRAM
CS
WE
PE
Row select
Col select
Element select

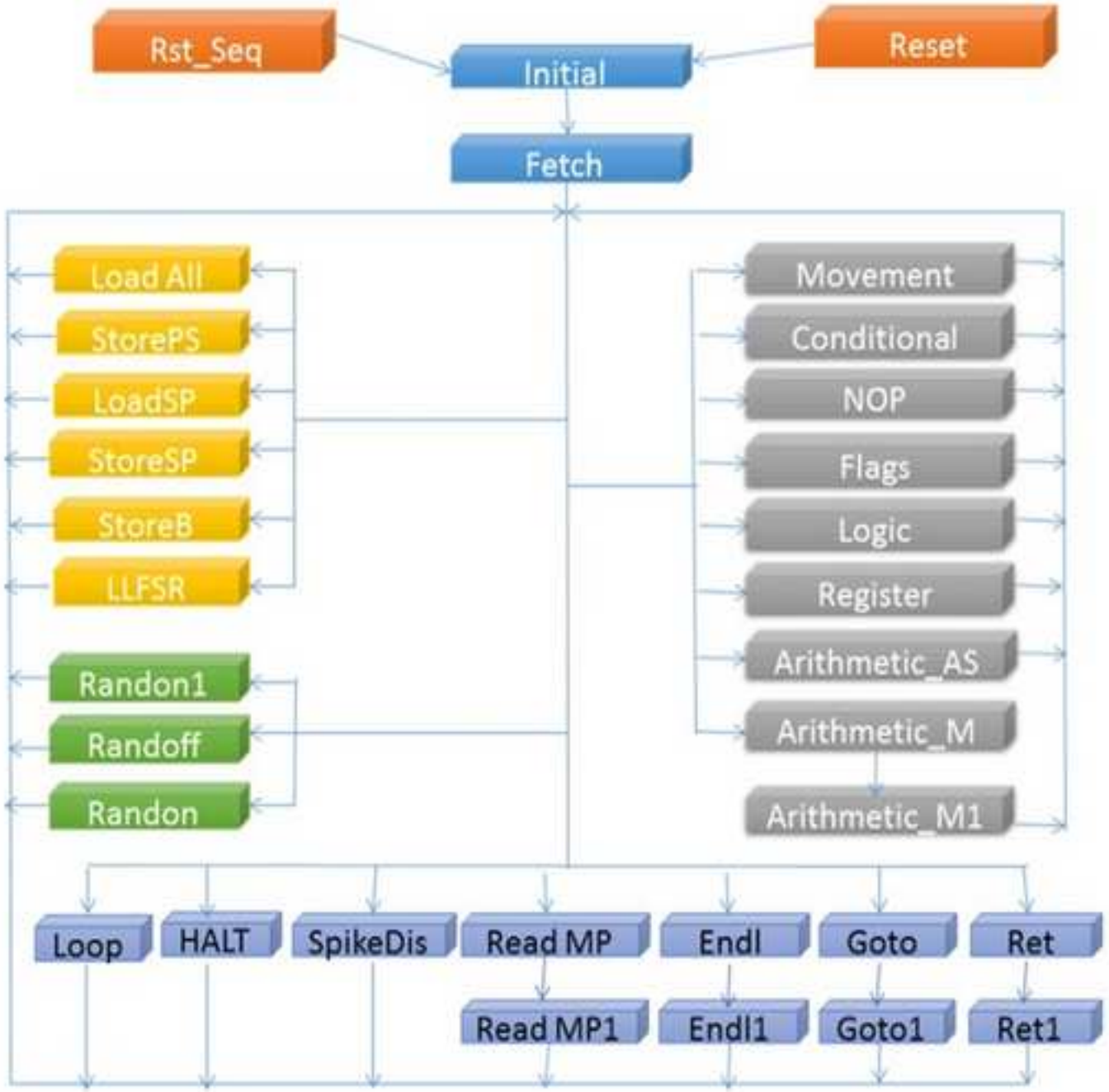
Figure


CPE
BRAM neu out
Data from Sequencer
Source Destination
BRAM syn out
LFSR
R1
R7
SR0
SR7
Shadow registers
Opcode
Source Dest
R0 (ACC)
BRAM neu in
BRAM syn in
ALU
Disable ALU
ALU Data Out
Buffer
M buffer

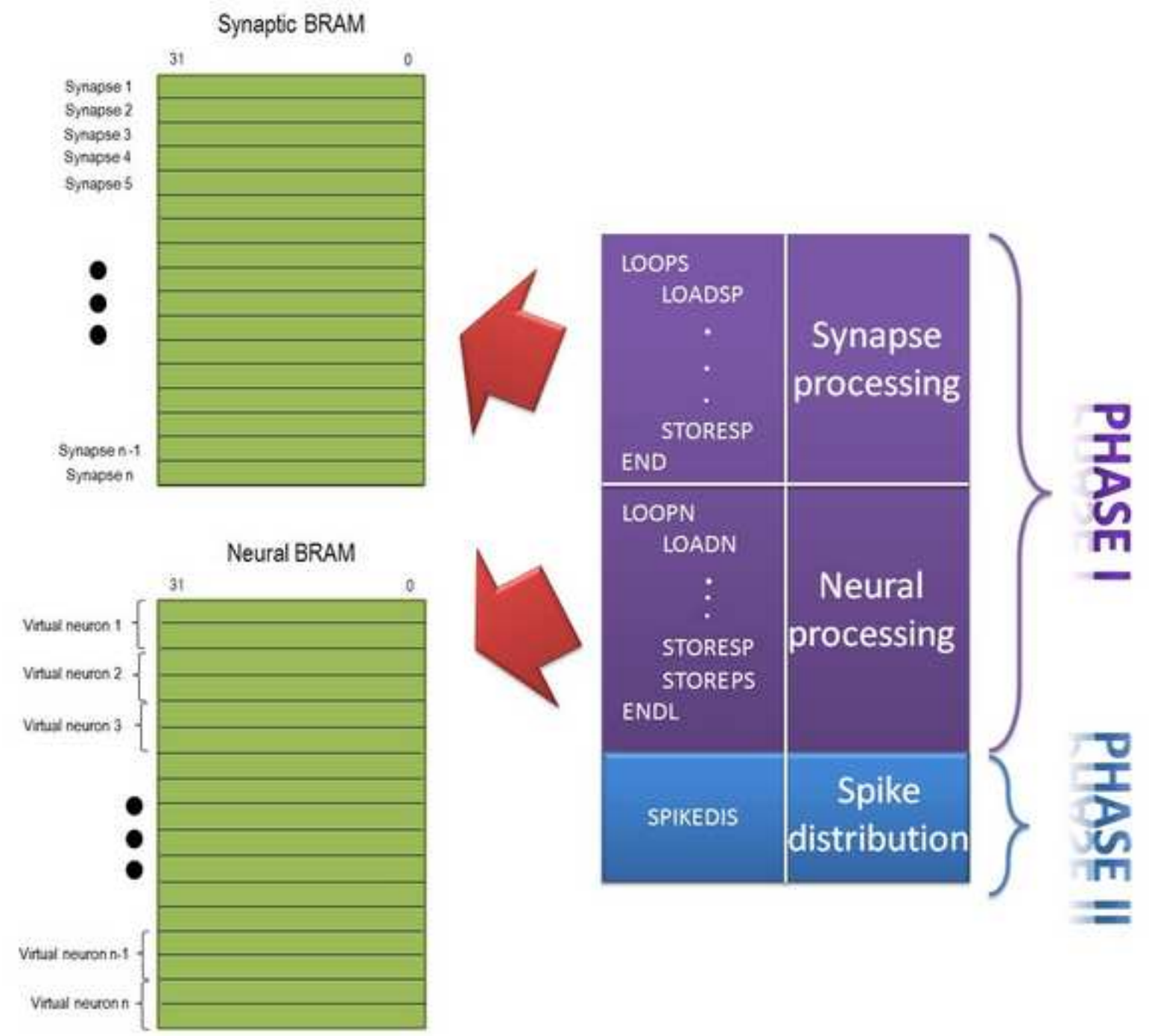
Figure


Sequencer
Zero Reg
DMEM
+
Data_alu_in
Control Signals
OPCODE
BRAM INSTRUCTION AND CONSTANTS
Count to Synaptic/Neural BRAM
+
IMEMP
PC
0
7
PC_Buffer (LIFO)
0
7
PC_LIFO
0
7
Loop 2 (LIFO)
0
7
Loop (LIFO)

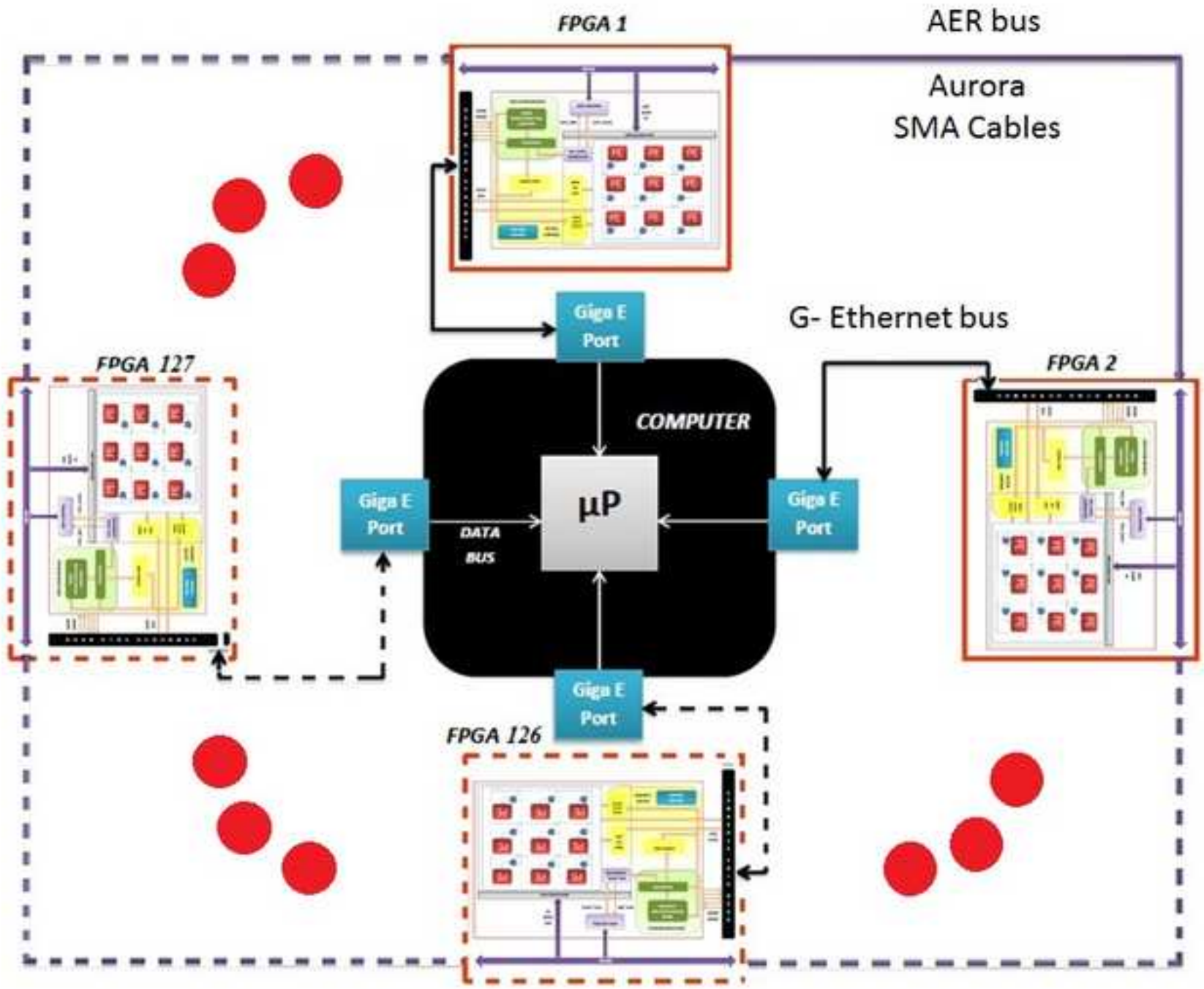
Figure


Rst_Seq
Reset
Initial
Fetch
Load All
StorePS
LoadSP
StoreSP
StoreB
LLFSR
Randon1
Randoff
Randon
Movement
Conditional
NOP
Flags
Logic
Register
Arithmetic_AS
Arithmetic_M
Arithmetic_M1
Loop
HALT
SpikeDis
Read MP
Endl
Goto
Ret
Read MP1
Endl1
Goto1
Ret1

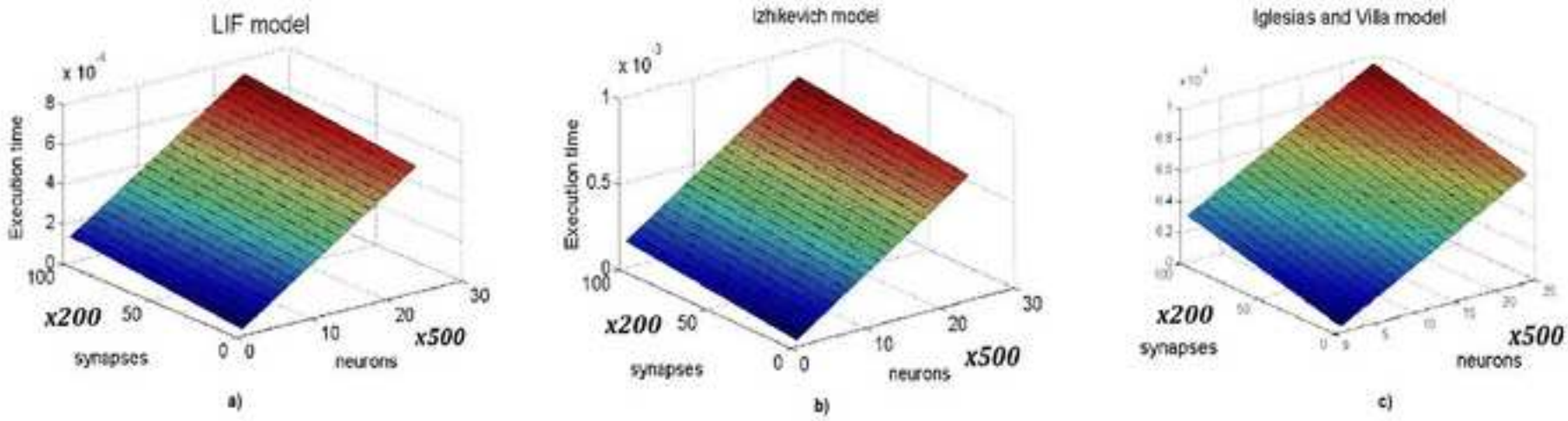
Figure


Synaptic BRAM
31
0
Synapse 1
Synapse 2
Synapse 3
Synapse 4
Synapse 5
Synapse n -1
Synapse n
Neural BRAM
31
0
Virtual neuron 1
Virtual neuron 2
Virtual neuron 3
Virtual neuron n-1
Virtual neuron n
LOOPS
LOADSP
STORESP
END
Synapse processing
LOOPN
LOADN
STORESP
STOREPS
ENDL
Neural processing
SPIKEDIS
Spike distribution
PHASE I
PHASE II

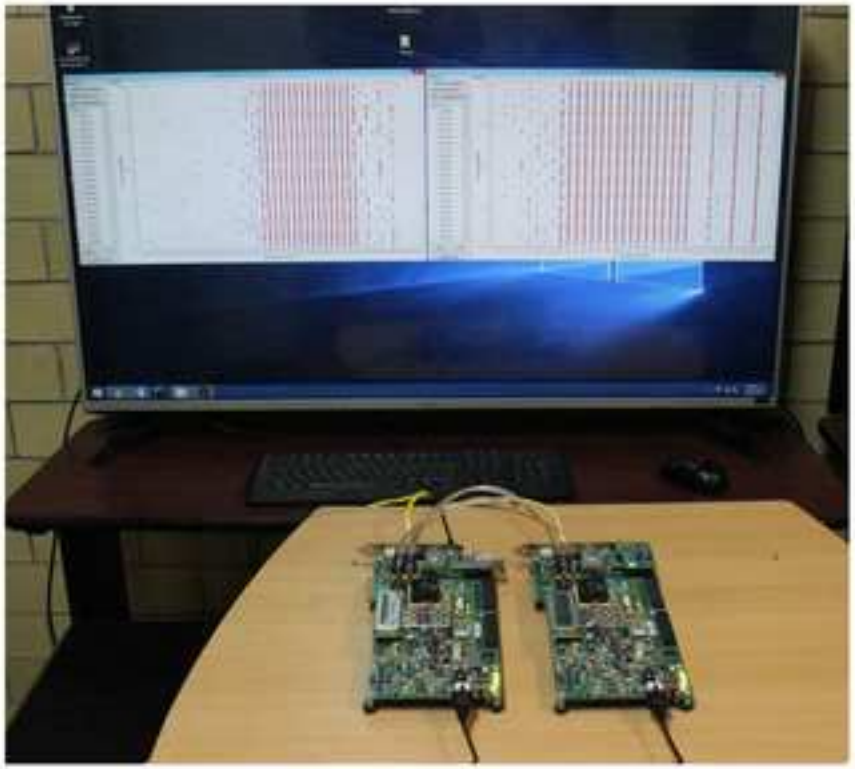
Figure


FPGA 1
AER bus
Aurora
SMA Cables
Giga E Port
G- Ethernet bus
FPGA 127
FPGA 2
COMPUTER
µP
Giga E Port
DATA BUS
Giga E Port
Giga E Port
FPGA 126

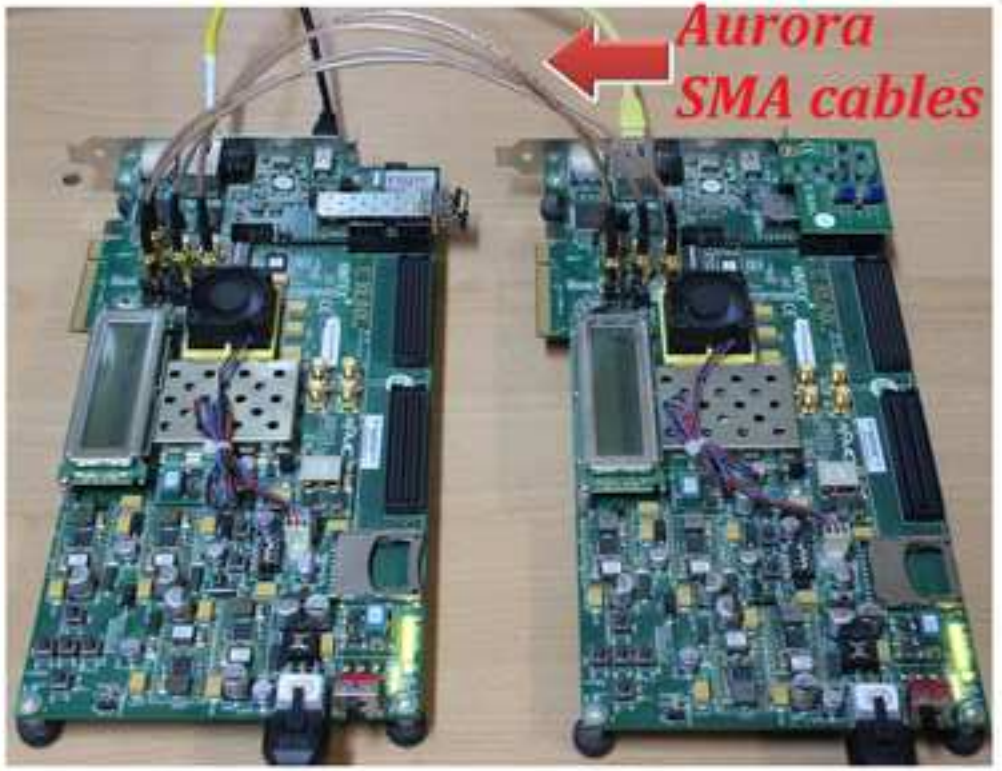
Figure


LIF model
x 10^-4
Execution time
8
6
4
2
0
100
x200
50
0
synapses
0
10
20
30
x500
neurons
a)
Izhikevich model
x 10^-3
Execution time
1
0.5
0
100
x200
50
0
synapses
0
10
20
30
x500
neurons
b)
Iglesias and Villa model
synapses
neurons
c)

Figure

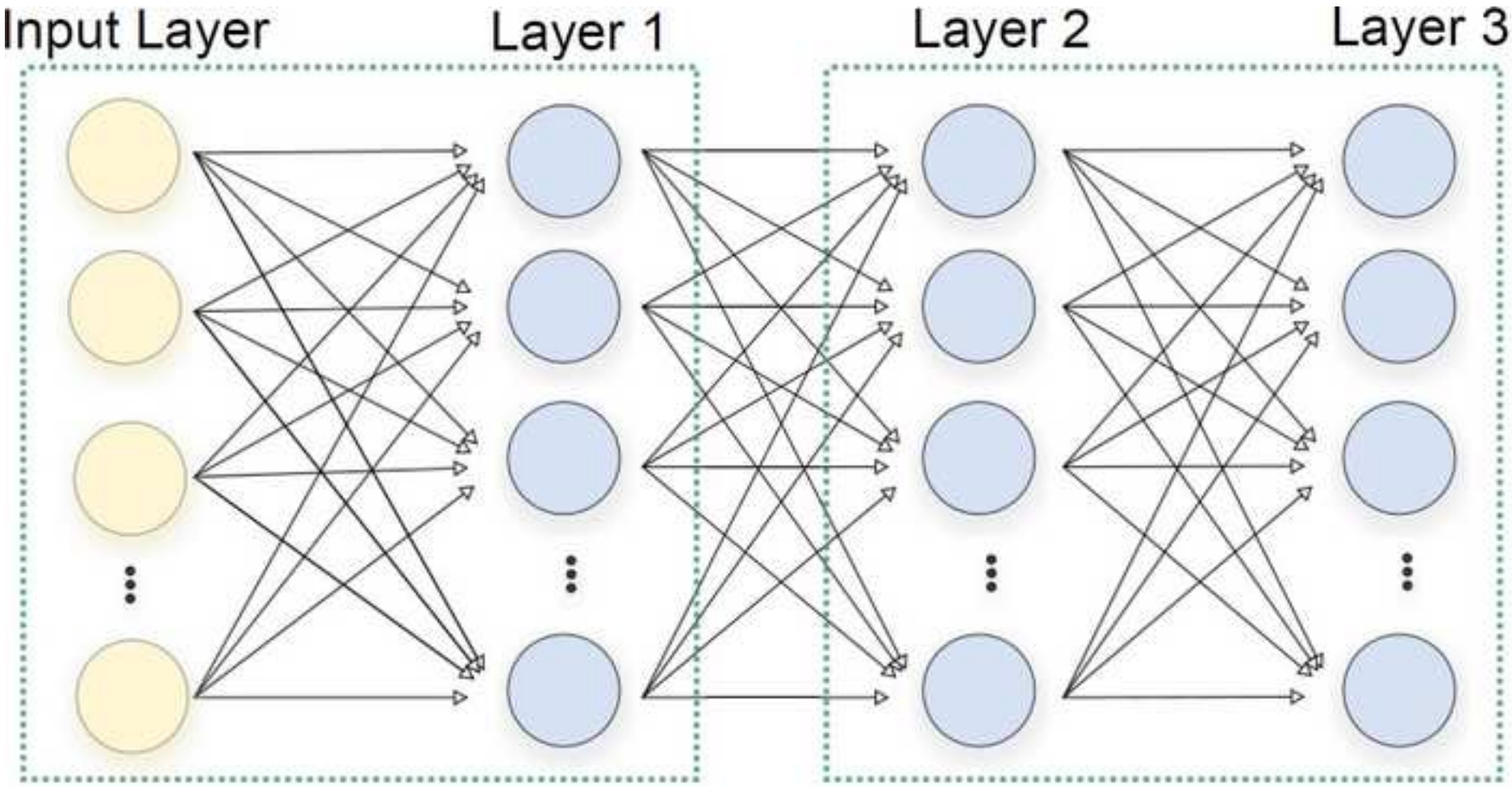


a)

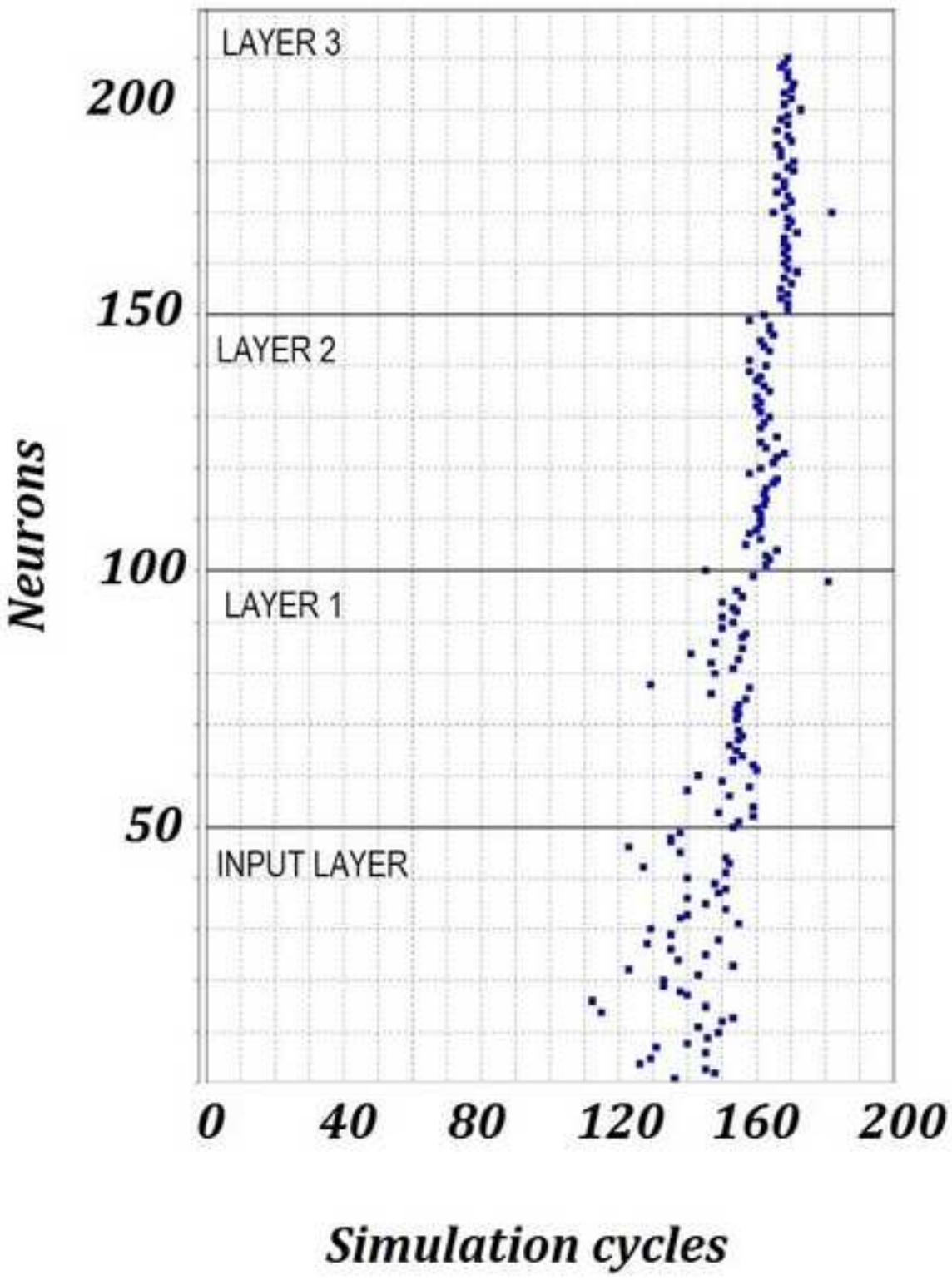


b)

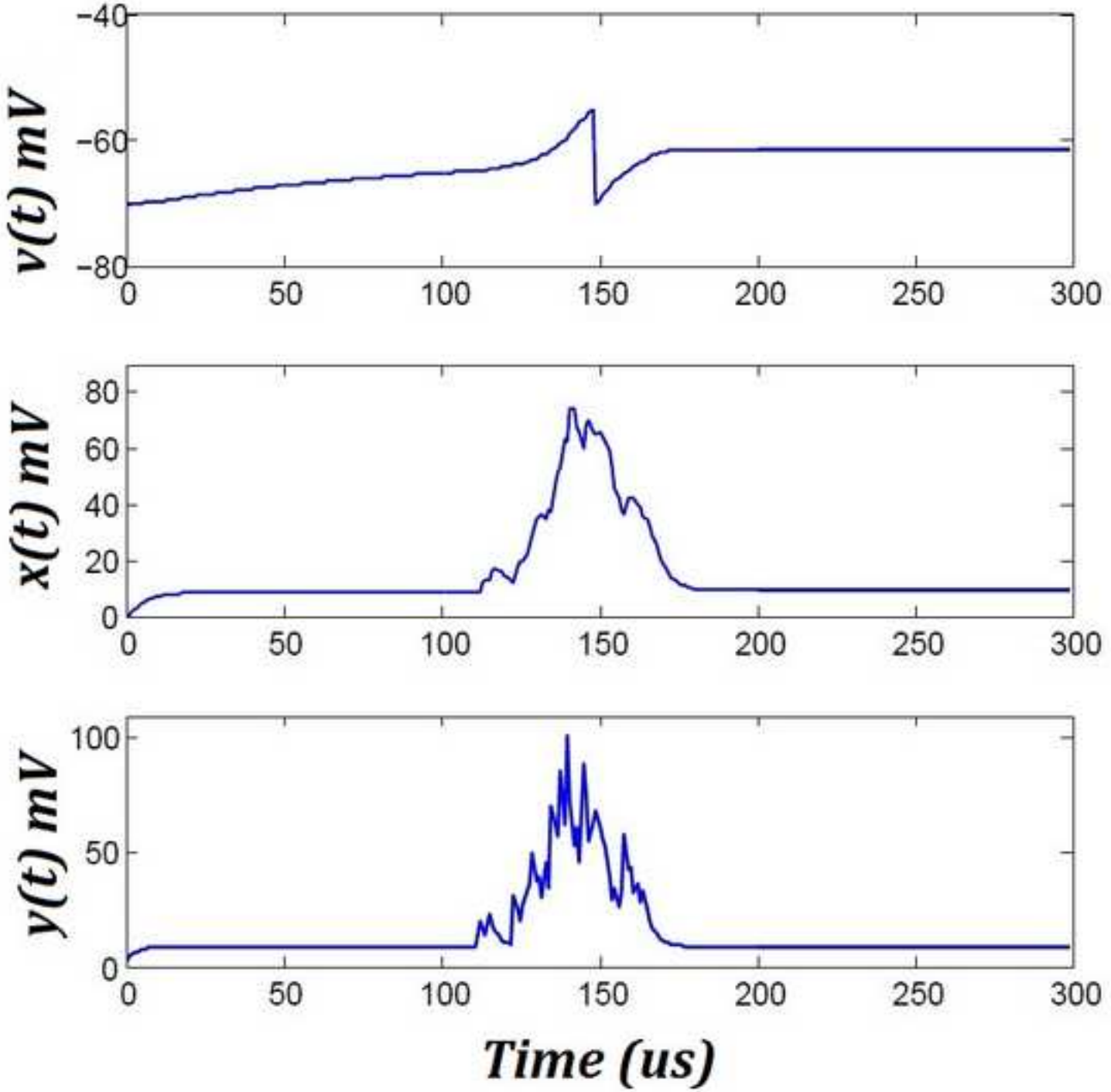
Figure



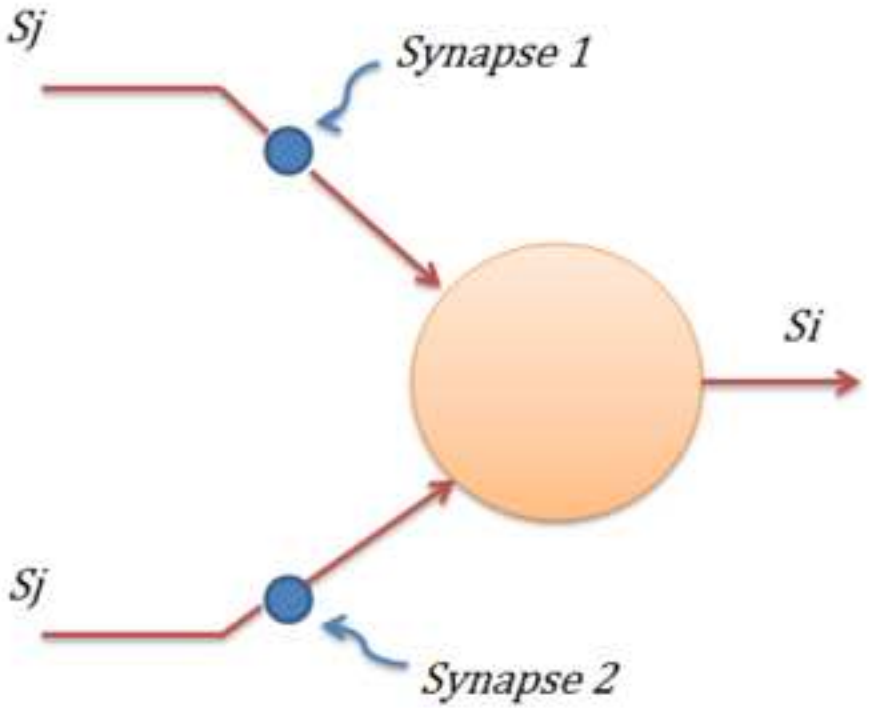
Figure


LAYER 3
LAYER 2
LAYER 1
INPUT LAYER
200
150
100
50
Neurons
0
40
80
120
160
200
Simulation cycles

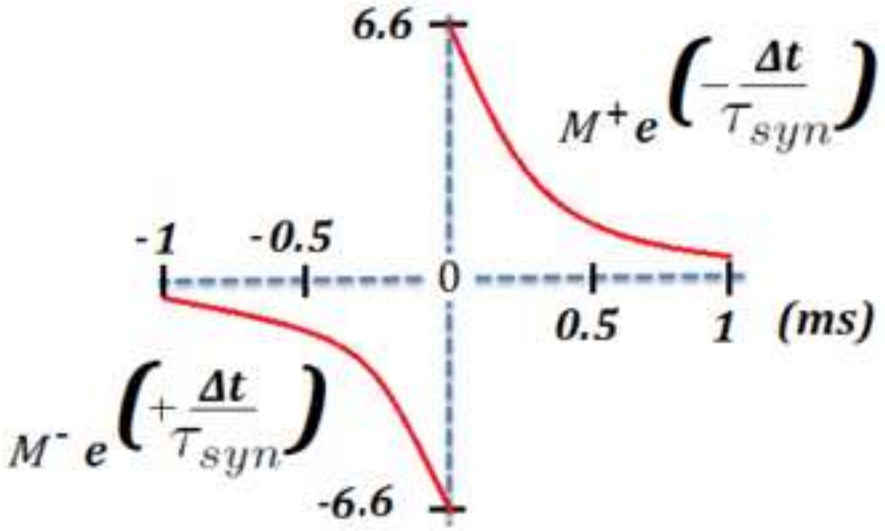
Figure


v(t) mV
−40
−60
−80
0
50
100
150
200
250
300
x(t) mV
80
60
40
20
0
y(t) mV
100
50
Time (us)

Figure

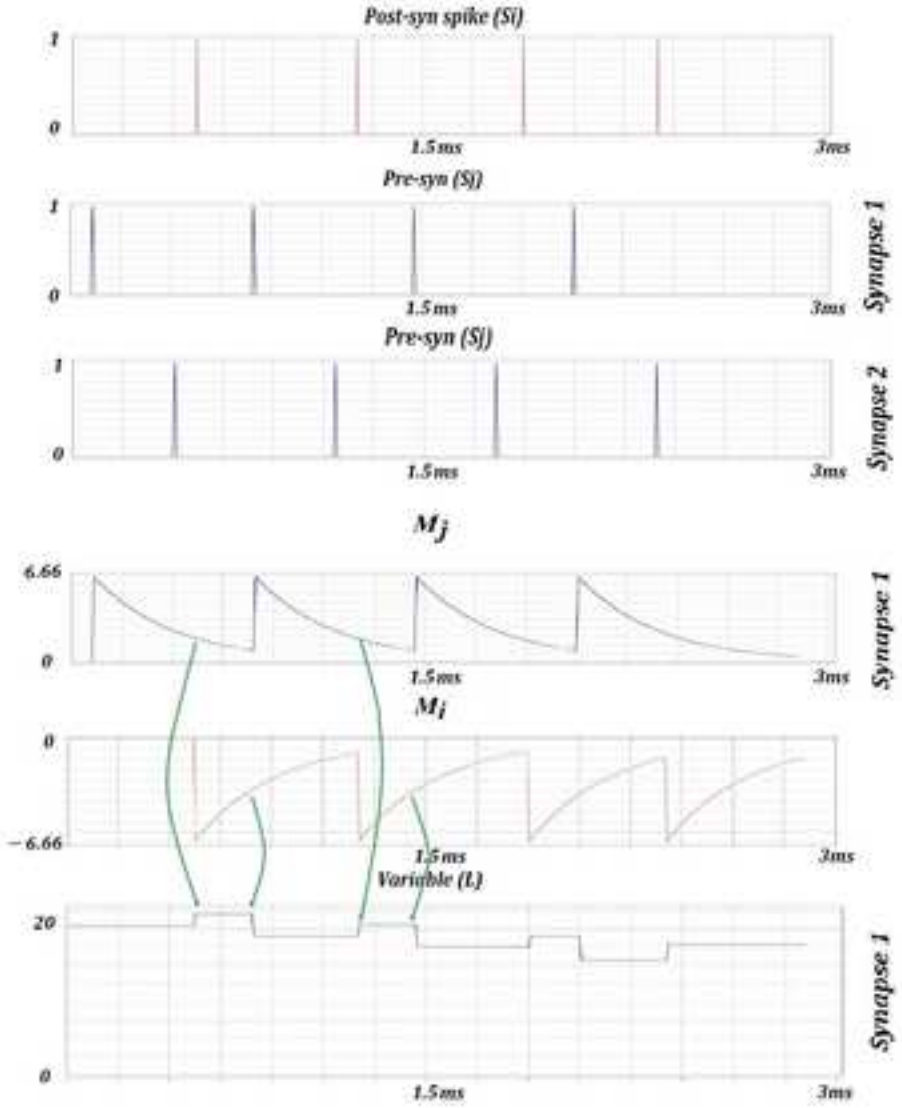

Sj
Synapse 1
Si
Sj
Synapse 2

a)

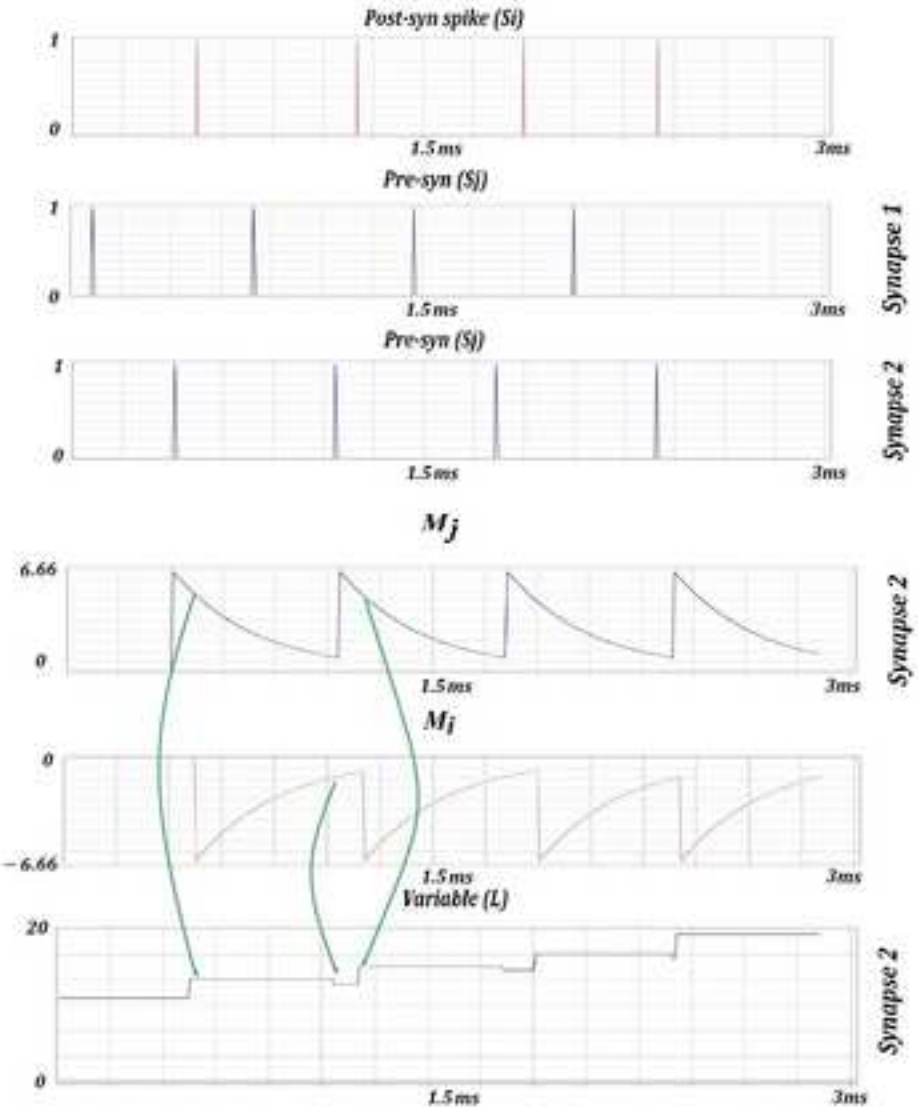

6.6
$M^+ e^{\left(-\frac{\Delta t}{\tau_{syn}}\right)}$
-1
-0.5
0
0.5
1
(ms)
$M^- e^{\left(+\frac{\Delta t}{\tau_{syn}}\right)}$
-6.6

b)

Figure

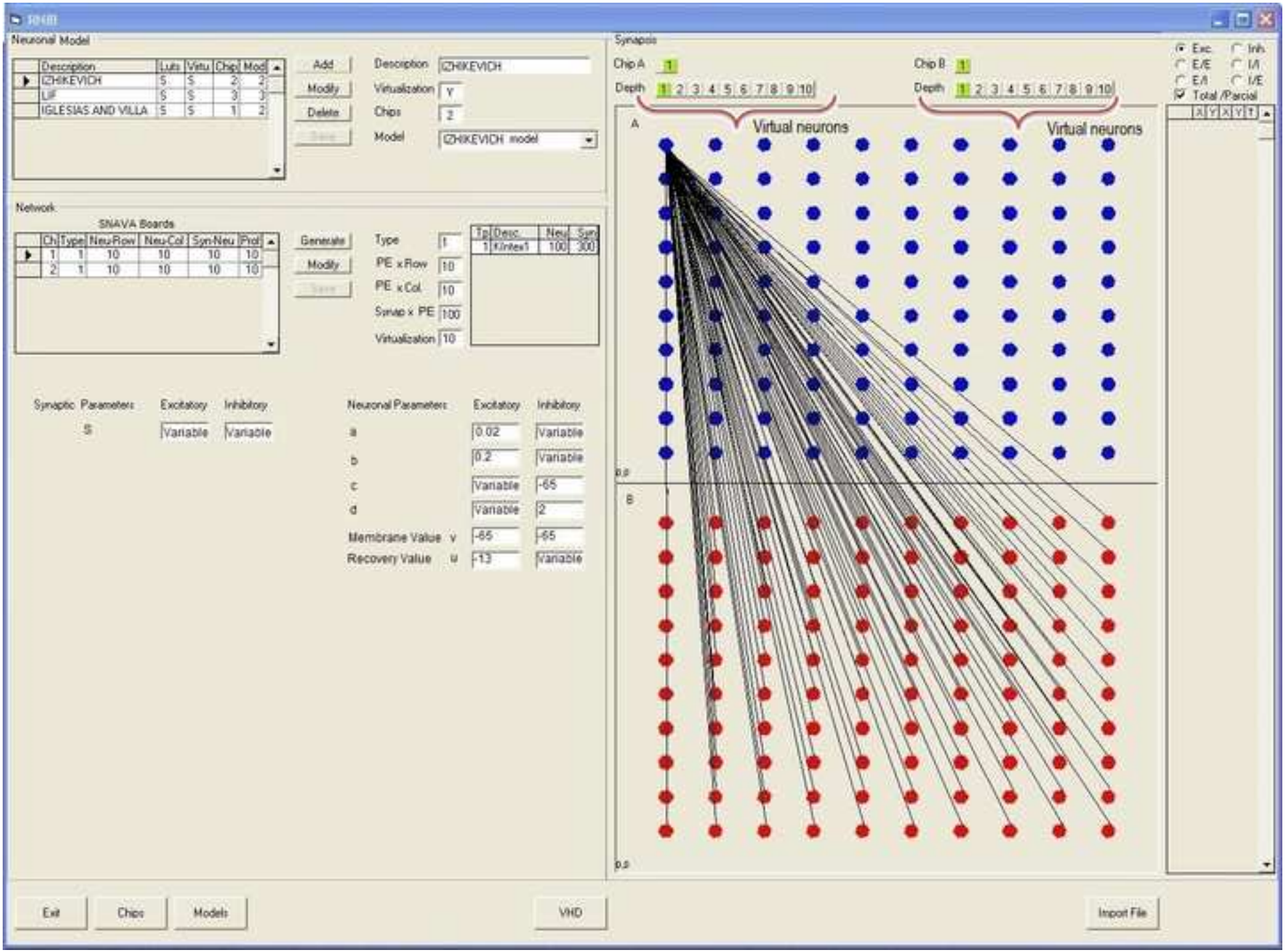


Synapse 1

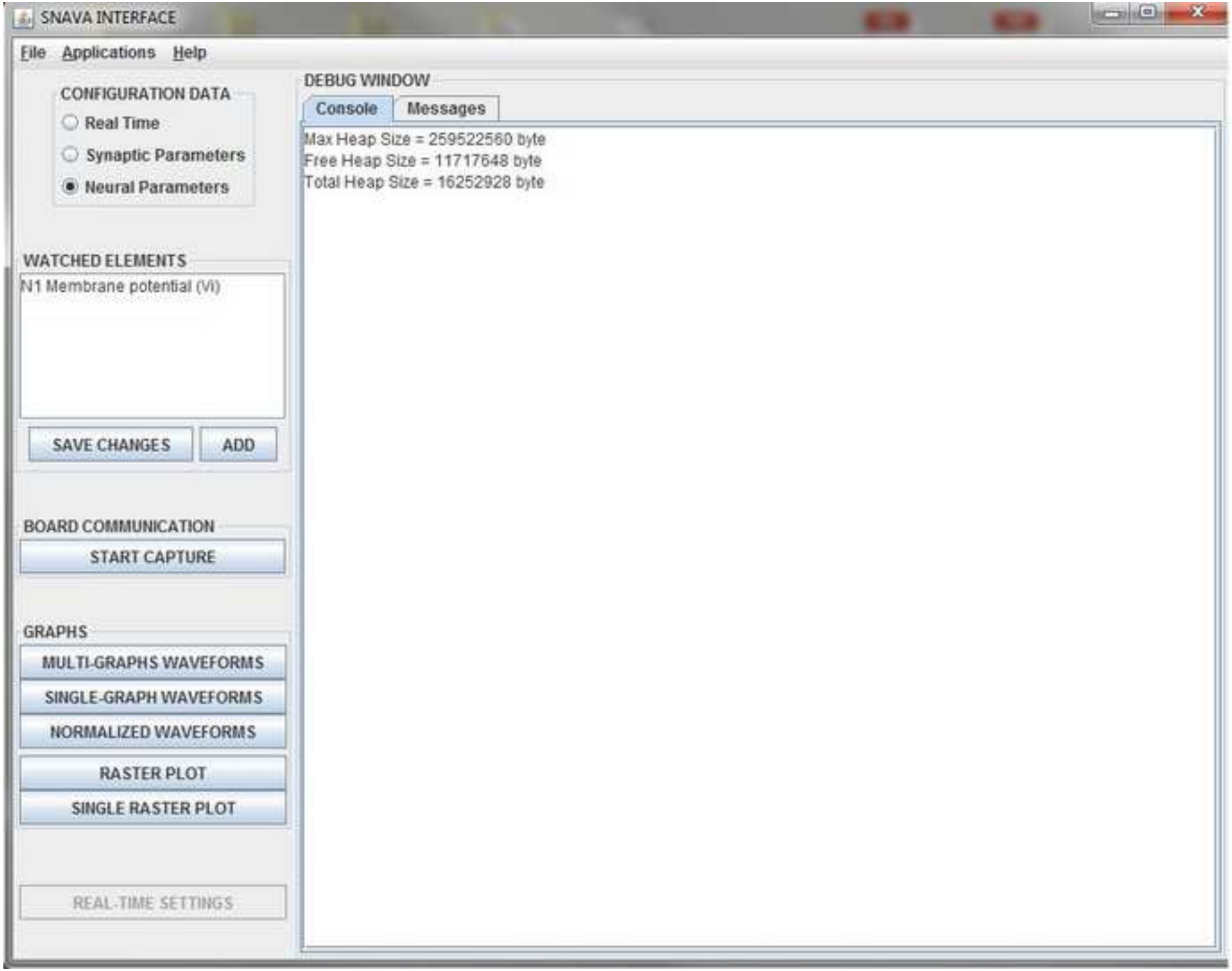


Synapse 2

Figure


Neuronal Model
Description
Luis
Virtu
Chips
Mod
IZHIKEVICH
LIF
IGLESIAS AND VILLA
Add
Modify
Delete
Save
Description
IZHIKEVICH
Virtualization
Y
Chips
2
Model
IZHIKEVICH model
Network
SNAVA Boards
Ch
Type
Neu-Row
Neu-Col
Syn-Neu
Prof
Generate
Modify
Save
Type
PE x Row
PE x Col
Synap x PE
Virtualization
Tp
Desc
Neu
Syn
Synaptic Parameters
Excitatory
Inhibitory
S
Variable
Variable
Neuronal Parameters
Excitatory
Inhibitory
a
0.02
Variable
b
0.2
Variable
c
Variable
-65
d
Variable
2
Membrane Value v
-65
-65
Recovery Value u
-13
Variable
Synapsis
Chip A
Chip B
Depth
1 2 3 4 5 6 7 8 9 10
Virtual neurons
Virtual neurons
A
B
Exc
Inh
E/E
I/I
E/I
I/E
Total /Parcial
X Y X Y T
Exit
Chips
Models
VHD
Import File

Figure


SNAVA INTERFACE
File
Applications
Help
CONFIGURATION DATA
Real Time
Synaptic Parameters
Neural Parameters
WATCHED ELEMENTS
N1 Membrane potential (Vi)
SAVE CHANGES
ADD
BOARD COMMUNICATION
START CAPTURE
GRAPHS
MULTI-GRAPHS WAVEFORMS
SINGLE-GRAPH WAVEFORMS
NORMALIZED WAVEFORMS
RASTER PLOT
SINGLE RASTER PLOT
REAL-TIME SETTINGS
DEBUG WINDOW
Console
Messages
Max Heap Size = 259522560 byte
Free Heap Size = 11717648 byte
Total Heap Size = 16252928 byte

LaTeX Souce Files

Click here to download LaTeX Souce Files: SNAVA - paper final version 1.tex